I.26r
no. 921
cop 2

UIUCDCS-R-78-921

UILU-ENG 78 1743
COO-2383-0047

SIMULATION OF CONTINUOUS NETWORKS WITH MODEL

by

Mitchell G. Roth
Thomas F. Runge

December 1978

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-78-921

SIMULATION OF CONTINUOUS NETWORKS WITH MODEL

by

Mitchell G. Roth
Thomas F. Runge

December 1978

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

Supported in part by the U. S. Department of Energy under contract
US ENERGY/EY-76-S-02-2383.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The MODEL simulation package was developed by a research group under the direction of Professor C. William Gear in the Department of Computer Science at the University of Illinois at Urbana-Champaign. Dr. Gear's broad expertise in the fields of computer science and mathematics provided inspiration and unification for the various research efforts that have been combined in the MODEL package. Other important contributions were made by the following individuals: Dennis Gannon developed the Cyber version of the MODEL compiler; Virginia Lo completed the graphical modeling program (DRAGON) and interfaced it to MODEL compiler; and Alex Orailoglu implemented the numerical routines for obtaining the steady state solution. The authors are especially indebted to Barbara Armstrong, who assiduously applied her considerable amanuensic talents to the task of producing this manual using the NROFF text formatting program on a PDP 11/34 computer.

M. G. R.
T. F. R.

0. INTRODUCTION

This manual is intended for the student, scientist, or engineer whose interests involve the solution of ordinary differential equations (ODEs). In many cases, such equations arise in connection with the development of computer simulation models. MODEL (Modular Ordinary Differential Equation Language) [1,2] is a very general and easy to use package for the formulation and simulation of continuous system models based on ordinary differential and nondifferential equations and generalized network (i.e., connective) effects. Partial differential equations (PDEs) can also be handled when they are transformed to a system of ODEs by the finite element method or the method of lines. MODEL combines both equation-oriented and network-oriented modeling features with a stiffly-stable implicit integration method [9,10], allowing it to be applied more widely and easily than any other continuous simulation package.

Other continuous simulation packages can be classified as either general purpose, equation-oriented packages in which models consist of user-formulated equations (e.g., CSSL-III [3]) or special purpose, network-oriented packages in which models consist of interconnected, predefined network elements (e.g., ECAP-II [4]). The equation-oriented packages require an explicit formulation of the equations of the model,

$$\underline{y}' = \underline{F}(\underline{y},t).$$

This restriction makes the application of equation-oriented packages impractical for all but the simplest network models because the amount of user analysis required to produce the explicit differential equations from a modestly complex network description is prohibitive. In fact, explicit formulation is an impossible task for a general nonlinear network. Thus, network-oriented packages are a great convenience where applicable. Unfortunately, these packages are usually limited to applications within a narrow range of disciplines. Digital analog simulators [5] are network-oriented packages intended for general application, but the complex circuits required to represent differential equations make them impractical for modeling general network elements.

Network-oriented packages include elements such as transistors which are described by sets of differential and nondifferential equations. However, modeling is restricted to a limited family of network elements. These packages provide no high-level features for equation-oriented modeling. This lack of generality also applies to the connective effects which can be handled. Special purpose languages have predetermined variables associated with network connections. Connections between elements in an electrical network package transmit single voltages and currents while connections in a structural network package transmit six components of displacement and six components of force. If a user wishes to apply one of these packages to a different discipline, or to an extension of the intended discipline not envisaged by the language designers, it is necessary to spend a great deal of time, either in rewriting the package to accommodate the model or in translating the model into a form acceptable to the package.

The purpose of MODEL (Modular Ordinary Differential Equation Language) is to make simulation of network models from a great many disciplines possible within a single framework, so that new concepts in the system can be incorporated into the model with ease, and so that interdisciplinary models (e.g., a biological model connected to an electrical network) can be handled directly. General network elements can be defined by sets of implicitly formulated differential and nondifferential equations and generalized network effects can be defined once and then applied automatically throughout a model. The use of an implicit integration method allows the simulation of a broader class of systems than can be handled with explicit integration methods.

The simulation of a system with the MODEL package consists of three steps: model formulation, model analysis, and numerical simulation. This manual is primarily concerned with the formulation of the simulation model. This model consists of a set of MODEL statements which can be entered into a filing system or punched on cards. After the model has been formulated, the remaining steps are performed by the two software processes of the MODEL package: the model analysis program, MAP; and the numerical simulation program, NSP.

MAP analyzes the model, simplifies it mathematically, and reformulates it for numerical simulation. The output of MAP is a FORTRAN run control program and a set of FORTRAN subroutines which represent the model in the form required by NSP. If desired, the FORTRAN code can be saved and inspected by the user before NSP is invoked. Saving this code allows the numerical simulation to be repeated many times without rerunning MAP. In addition, a user who is familiar with FORTRAN can modify the code produced by MAP and/or interface his own FORTRAN subroutines to the numerical simulation at this point.

When the model analysis has been completed, NSP is invoked to compute a numerical approximation to the solution of the model equations. The output of NSP is a solution history for the model which can be printed, plotted, or saved for additional computer analysis.

The organization of the manual parallels the development of a simulation model. Chapter 1 discusses the use of ODEs to model physical systems. Chapter 2 introduces the MODEL language, which is used to represent the ODEs on a computer. For physical systems consisting of one or more interconnected subsystems, MODEL allows the equations to be organized by subsystems and then connected in any desired manner to form a network model, as described in Chapter 3. The run control program sets a number of variables which control NSP. Chapter 4 describes the manner in which the run control program is generated and the uses of the control variables. Chapter 5 explains the diagnostic information produced by MAP and NSP when errors are detected. It also discusses the correspondence between the MODEL variables and the FORTRAN variables used by NSP.

Several appendices are also included. They summarize the MODEL language and provide examples of a simulation model and typical job control language statements. Appendix A presents the various MODEL language statements in a concise list which can be used as an index and ready reference to the detailed presentation of Chapter 2 and 3. This appendix also describes the order of statements in a model definition. Appendix B shows the complete formulation of two example models and selected portions of the output produced during the analysis and simulation of these models. Appendix C shows the job control used to invoke MAP and NSP. The job control will, of course, be different for other machines but Appendix C provides examples for users who are familiar with IBM 360 JCL and Cyber NOS. Appendix D lists the various error messages which may be produced during model analysis and gives an indication of the cause, and possible corrective actions for each.

The MAP and NSP processes have been implemented in FORTRAN to allow their use on a variety of large computers. The MODEL package is presently running on IBM 360/75 and CDC Cyber 175 machines.

In addition to the MAP and NSP processes, MODEL users at certain installations will have access to an interactive graphical modeling process called DRAGON. DRAGON allows the user to construct models interactively at a remote CRT terminal using graphical representations of network elements and connections, and to add to and select from a library of predefined network elements. When the construction of a model is completed, DRAGON reformulates the graphical model into MODEL language text statements for processing by MAP.

The DRAGON process is implemented in the Graphical Interactive Machine Language (GIML) developed at the University of Illinois. GIML runs interpretively on a PDP-11 minicomputer under the Bell Laboratories UNIX operating system. To use GIML on other systems a GIML interpreter must be programmed to emulate a hypothetical machine whose instruction set was designed to facilitate interactive graphics.

Since GIML is not as portable as FORTRAN, the DRAGON process will not be available at many installations. Therefore, this manual will not describe the use of DRAGON to construct system models. Interested users are referred to the DRAGON User's Manual [6] for information about the graphical modeling techniques that are available. GIML and the GIML interpreter are described in detail in [7] and [8], respectively.

1. MODELING WITH ORDINARY DIFFERENTIAL EQUATIONS

The mathematical formalism of ordinary differential equations (ODEs) is the single most powerful tool for the description of dynamic phenomena. Any system whose rate of change depends only on the current state of the system can be described by an ODE. Owing to their generality, ODEs have been applied in every discipline that involves quantitative measurement and prediction. This chapter provides a brief introduction to the use of ODEs in simulation models.

1.1. Formulating the MODEL Equations

In many problems, the quantities under study vary continuously as an independent variable changes. Commonly, time is the independent variable, and the problem is characterized at any time, t, by a finite number of quantities, which form a time dependent state vector $\underline{x}(t)$. The rate of change of the state vector is denoted by the derivative of $\underline{x}(t)$ with respect to t, which is written as $\underline{x}'(t)$ or $d\underline{x}/dt$. When $\underline{x}'(t)$ can be expressed as a vector function of $\underline{x}(t)$ and t, we are led to the first order linear ODE,

$$\underline{x}'(t) = \underline{f}(\underline{x}(t),t). \tag{1.1}$$

The solution to ODE (1.1) is a vector function $\underline{x}(t)$ whose derivative satisfies the ODE. When an initial state vector, $\underline{x}(t0)$, has been specified at t0, the starting value of the independent variable, it can be shown that ODE (1.1) has a unique solution under very mild conditions that ensure that $\underline{f}(\underline{x},t)$ is reasonably well behaved.

More generally, it is not always possible to write an explicit equation for $\underline{x}'(t)$. In such cases, $\underline{x}'(t)$ is implicitly determined by an ODE of the form

$$\underline{F}(\underline{x}'(t),\underline{x}(t),t) = 0. \tag{1.2}$$

In the implicit formulation, it is more difficult to determine if the ODE defines a unique solution. A sufficient condition for uniqueness is that $\underline{F}(\underline{x}',\underline{x},t)$ be continuous and nonsingular with respect to $\underline{x}'$. A system of n equations in n variables, written as $\underline{g}(\underline{x}) = 0$, is nonsingular with respect to $\underline{x}$ if the determinant of the Jacobian matrix $d\underline{g}/d\underline{x}$, is nonzero. If the number of equations does not equal the number of variables, the system is always singular. However, this condition is unnecessarily restrictive because we wish to include nondifferential equations and allow the number of variables to differ from the number of equations. Both conditions would automatically cause $\underline{F}(\underline{x}',\underline{x},t)$ to be singular with respect to the derivative vector. The necessary condition for a unique solution is that the components of $\underline{x}$ which are differentiated must uniquely determine the derivatives in the implicit ODE. In this case, the initial state may be completely specified by a subspace of the $\underline{x}$ vector. The implicit ODE is functionally equivalent to an explicit ODE and therefore has a unique solution.

The solutions to ODEs (1.1) and (1.2) can be found analyticaly only when $\underline{f}(\underline{x},t)$ and $\underline{F}(\underline{x}',\underline{x},t)$ are relatively simple functions. For many scientific and engineering problems, it is necessary to approximate the exact solution by a computational procedure. The first computer programs for solving ODEs could only solve explicit equations in the form of ODE (1.1). This restriction was necessary in order to represent the equations by FORTRAN style arithmetic. MODEL removes this restriction and allows equations to be written as

$$\underline{g}(\underline{x}',\underline{x},t) = \underline{h}(\underline{x}',\underline{x},t), \tag{1.3}$$

where g and h are arbitrary expressions involving constants, parameters, variables, and built-in and user supplied operators and functions (the dependence of $\underline{x}$ and $\underline{x}'$ on t is not shown explicitly because there can be only one independent variable; all other variables are assumed to be dependent).

Equations in MODEL represent true mathematical equalities, as opposed to simple arithmetic assignment statements. When a system of equations is being solved, the order in which the equations appear has no importance, since the solution must simultaneously satisfy all of the equations. MODEL achieves this generality by rewriting all equations in the implicit form

$$\underline{F}(\underline{x}',\underline{x},t) = \underline{g}(\underline{x}',\underline{x},t) - \underline{h}(\underline{x}',\underline{x},t) = 0 \tag{1.4}$$

The stiffly-stable integration method which MODEL uses employs an implicit approximation to the solution of ODE (1.4). The same method may also be used to solve nondifferential implicit equations of the form

$$\underline{g}(\underline{x},t) = 0, \tag{1.5}$$

which is really a zero order implicit ODE.

Thus, the complete set of MODEL equations may include both differential and nondifferential equalities. For problems which involve networks of interconnected elements, MODEL exploits this capability by allowing arbitrary types of connection terminals to be defined. When two elements are connected, certain relationships are established by virtue of the connection. In an electrical circuit a connection represents a point at which the voltages of the connected elements are equal. In a mechanical structure, a connection point must equate six components each of displacement and force. Both situations can be handled by automatically inserting the necessary equations corresponding to each connection. The ability to solve simultaneous implicit systems of equations obviates the former need to reorder the equations and variables in order to obtain an explicit system of equations.

As a simple example of the manner in which MODEL equations are formulated, consider the differential equations for the motion of a projectile, launched vertically from the surface of the earth when air resistance is neglected, and gravitational acceleration is assumed to be constant. The independent variable here is time. Therefore, let $x(t)$ be the height of the object at time t, and assume it is launched with velocity, v. According to Newton's Laws,

$$x''(t) = -g, \tag{1.6}$$

$$\text{where} \quad \begin{aligned} x(0) &= 0, \\ x'(0) &= v, \end{aligned}$$

and g is the constant acceleration due to gravity.

This is an example of a second order differential equation. The second derivative must be rewritten in terms of the first derivative of an auxiliary variable, s, which represents the velocity of the projectile. The result is a first-order system of two equations:

$$\begin{aligned} x' &= s \\ s' &= -g, \end{aligned} \tag{1.7}$$

$$\text{where} \quad \begin{aligned} x(0) &= 0, \\ s(0) &= v. \end{aligned}$$

This represents an acceptable form of equations in MODEL. Alternatively, the above equations could be written as:

$$\begin{aligned} x' - s &= 0 \\ s' + g &= 0, \end{aligned} \qquad (1.8)$$

$$\text{where} \quad \begin{aligned} x(0) &= 0, \\ s(0) &= v. \end{aligned}$$

In fact, the above transformation in which one side of each equation is zero is employed by MODEL to obtain the implicit equations in standardized form.

Most physical systems are assumed to exhibit a predictable behavior that depends on the initial state of the system. The equations that model such a system should also possess this property. If the number of variables (dimension of x vector) exceeds the number of equations (dimension of F), it is unlikely that the equations have a unique solution. On the other hand, if there are more equations than variables, the system can have a solution only if the equations are functionally dependent. Two functions $f(x,y)$ and $g(x,y)$ are dependent if there exists a nontrivial function, $h(u,v)$, such that $h(f(x,y),g(x,y)) = 0$. This implies that the system of equations $f(x,y) = 0$, $g(x,y) = 0$ may be transformed (by means of h) to the degenerate equation $0 = 0$. This may be generalized directly for systems of more than two equations. For any properly posed physical problem which can be described by ODEs it is possible to obtain a nonsingular system of equations which has a unique solution.

1.2. Steady-State Solution

Before the solution to an implicit ODE can be found, the values of all variables and derivatives must be known for a fixed starting value of the independent variable. Collectively, these initial values comprise the steady-state solution of the ODE. In the example above, the initial values x(0) and s(0) were explicitly specified at time zero. The initial derivatives can be obtained directly from (1.7) as

$$x'(0) = v, \quad s'(0) = -g. \tag{1.9}$$

In general, the steady-state solution consists of vectors $\underline{x}(t0)$ and $\underline{x}'(t0)$ which satisfy the implicit ODE

$$\underline{F}(\underline{x}',\underline{x},t0) = 0, \tag{1.10}$$

where t0 is the starting value of the independent variable. The initial state of the physical system determines certain components of the steady-state solution. The remaining components are adjusted by the numerical simulation program (NSP) until (1.10) is satisfied to within a specified tolerance, which is set in the run control program (Chapter 4).

The steady-state solution is usually employed to initialize the transient solution (section 1.3) and this is the default mode of analysis. Some problems will only require a steady-state solution. For example, the solution of the linear system of equations

$$A\underline{x} = \underline{b}, \tag{1.11}$$

where $\underline{x}$ and $\underline{b}$ are vectors and A is a matrix, is not time dependent when A and $\underline{b}$ are constants. This type of problem arises in the DC analysis of electrical circuits and the static analysis of structural frameworks. The appropriate mode of analysis can be selected in the run control program.

Some attention must be given to the manner in which the initial conditions are specified. If too few initial conditions are supplied, the steady-state solution may not be unique. In terms of the physical system, this means that the initial state is incompletely specified. When this situation arises, it usually indicates that the equations have been incorrectly or redundantly formulated.

It is also possible to overspecify the initial state, in which case there may be no steady-state solution. As a general rule, when the number of variables equals the number of equations, an initial value should be supplied for each differentiated variable. Then, the total number of uninitialized variables and derivatives will equal the number of equations. If the system of equations is continuously differentiable and nonsingular with respect to the uninitialized terms in some region of the solution space which includes a steady-state solution, then the solution will be unique throughout the region where the equations are nonsingular. If the equations are nonsingular over the entire solution space, then a unique steady-state solution can always be found. In practice, it is usually difficult to determine if a system of equations is nonsingular in a certain region. Thus, to some extent, it is necessary to rely on physical intuition in assigning initial values.

1.3. <u>Transient Solution</u>

The position and velocity of the projectile in (1.7) change with time. The motion of the object can be described by a time dependent function which solves the ODE. Such a function is called the transient solution of the ODE. In this case, the transient solution can be found analytically by integrating the ODE to obtain

```
x(t) = vt - 1/2gt**2,                          (1.12)
s(t) = v - gt.
```

The MODEL package computes an approximation to the transient solution by numerical integration. This procedure uses polynomials to approximate the solution to the ODE over small intervals in the independent variable. The intervals are called steps and the stepsize is chosen by NSP to satisfy the error tolerance that is specified for the solution in the run control program (Chapter 4). A smaller step produces a more accurate solution, but more steps will be required to complete the integration. Therefore, the required accuracy of the transient solution affects the computational cost of the solution.

1.4. Accuracy and Scaling

The units which are chosen for the variables in a simulation model can affect the accuracy of the numerical solution. While the arithmetic unit of the computer can handle numerical magnitudes over a very large range, the precision of the numerical representation is finite, as shown in Table 1.1.

	IBM	CDC
Largest magnitude	10**75	10**322
Smallest nonzero magnitude	10**(-78)	10**(-293)
Precision, decimal digits	16	29

Magnitude and Precision of Numerical Representation in MODEL Package for IBM 360 Series and CDC Cyber and 6000 Series Computers.

Table 1.1

If two numbers whose exponents differ by more than the precision of the computer are added, the smaller number will be totally lost from the result. The least error is introduced when operating on numbers of comparable magnitudes. The units for the dependent variables should therefore be selected to equalize the anticipated magnitudes of the various components of the solution as much as possible. For example, length may be measured in either microns or meters, but the magnitude of the length variable will be changed by a factor of one million. The units which are most appropriate for the problem should be chosen.

Since all of the terms in an equation must be dimensionally compatible, proper scaling of the variables will usually equalize the coefficients and physical constants which appear in the equations. This also helps to produce a computationally well conditioned problem.

The scaling of the dependent variables affects each variable individually. The scaling of the independent variable affects all derivatives in the model. The units for the independent variable should be chosen so that the magnitudes of the largest derivatives are of the same order as the largest variables. For example, suppose an electrical oscillator circuit has a period of one microsecond. The amplitude of the signal would probably be measured in volts. If time, the independent variable, is measured in seconds, the derivatives in the equations will be of order 10**6 volts per second. If the time unit is microseconds, on the other hand, the derivatives will have magnitudes of about 1 volt per microsecond, which is much closer to the magnitude of the signal amplitude.

The error tolerance that is specified for the transient solution in the run control program controls the relative error, i.e., the number of correct digits, in the solution for variables whose magnitude exceeds one. For variables whose absolute value is less than one, the error tolerance is used to control the absolute error in the solution. Effectively, this means that any variable whose magnitude is less than the error tolerance may have no significant digits. Thus, to guarantee that the specified relative error is achieved, variables should be scaled to have a magnitude greater than one at all times. However, this is not possible when the solution changes sign. The severity of the error that is introduced is dependent on the stability of the physical system at small values of the affected variable. If the influence of the variable is proportional to its magnitude, then any error associated with small magnitudes will be relatively insignificant and the absolute error criterion is appropriate. On the other hand, if the physical system has a singularity at zero for some variable, errors which occur at small magnitudes may be amplified enough to affect the accuracy of the entire solution. For example, gravitational force is inversely proportional to the square of the distance. In computing a satellite orbit, low altitude errors have a critical effect on the trajectory. In such cases, the magnitude of the variable should be biased away from zero to maintain the accuracy of the solution.

With optimal scaling the finite precision of the computer representation still limits the maximum accuracy of the numerical solution to the values shown in Table 1.1. A particular set of scale factors is optimal only for certain magnitudes of the variables. The range of magnitudes which a variable may assume without affecting the accuracy of the computation is also determined by the precision of the computer arithemetic. As a rule of thumb, if the magnitude of a linear variable changes by more digits than the precision of the computer arithmetic, then a significant computational error may occur. To minimize the error the problem should be scaled for highest accuracy in the range where it is most unstable. For example, zero stable variables should be scaled to control the maximum magnitude relative to the error tolerance. The largest relative errors will then be incurred at small magnitudes and have little effect on the accuracy of the solution.

1.5. <u>MODEL Structure</u>

The number of equations required to model a complex system may be very large. Some means of organizing the equations is therefore highly desirable. Most physical systems have a natural hierarchical structure. That is, they can be described at a series of levels of increasing complexity. At each level, various subsystems are combined to create the next higher level. A complex system can then be conceptualized in terms of a relatively small number of interconnected subsystems.

The MODEL language employs this type of organization for constructing network models. Such models consist of interconnected components called elements. Each element is defined in terms of equations or other elements. An example of a network system is an electrical circuit, which consists of circuit elements such as transistors, capacitors, batteries, etc.

In formulating a network model, it is necessary to decide where to draw the subsystem boundaries, i.e., what constitutes an element. A network simulation is usually used to test alternative arrangements or parameter values in the components of a system and the model elements are isomorphic to these components. During the development of a system, abstract concepts are transformed into concrete entities and the elements which originated as functional components evolve to represent physical components. The modular, hierarchical fashion in which network models are defined using MODEL is a conceptual aid in modeling complex systems, and it saves the bother of specifying multiple definitions for elements which occur several times in a system. It also facilitates the use of element definitions borrowed from other users or from a library. Since a change in the definition of one element does not usually necessitate changes in the definitions of other elements used in the same model, this model structure also facilitates the debugging of element definitions and the substitution of different definitions of the same element for different kinds of analyses. The simulation created by the MODEL package is therefore a powerful design tool that can be used at all stages of system development.

This chapter concludes with example network systems from several disciplines. Typical element and connection definitions are presented in each example to illustrate that an element may correspond to either physical or functional components.

Areas of application of the MODEL package include electrical, mechanical, structural, chemical, biological, economic, and fluid networks, to name a few. The use of these types of components in network models is illustrated by the examples below.

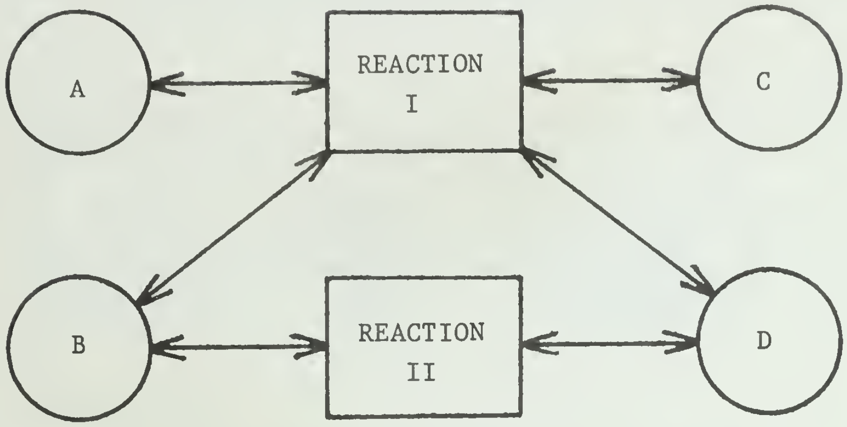


Figure 1.1. Elements in a Chemical Kinetics Network.

In Figure 1.1, the circles represent "containers" for different radicals present in the model. The gain and loss of radicals occurs along the connections to the rectangular reaction "boxes". The rate of flow of these radicals is governed by the reaction equation, written chemically as

$$A + B \underset{\leftarrow K2}{\overset{K1 \rightarrow}{=\!=\!=\!=\!=\!=}} C + D.$$

If A stands for the concentration of radical A (that is, the "amount" in the "container" for A), and similarly for B, C, and D, then the rate of gain of A and B from this reaction (and hence the rate of loss of C and D) is

$$(A + B)' = K2*C*D - K1*A*B.$$

This equation can be associated with the reaction element. The concentration of radical A is determined by a differential equation relating A' to the sum of all flows into and out of the A container. This equation can be associated with the container element. The connections must transmit the concentrations and rates of flow of radicals between elements.

Biomedical Cell

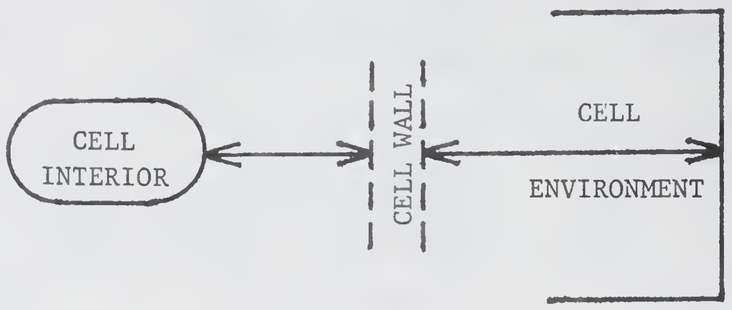


Figure 1.2. Biomedical Cell Modeled by Elements.

A cell might be modeled by three elements as shown in Figure 1.2. The elements would represent the concentrations of chemicals of interest, potential levels, etc., and would be modeled by equations. The connections would represent the flow of chemicals and information from one neighbor to another.

Hydraulic Flow (e.g., a refinery)

Types of elements that can arise include:

1. Storage tanks--modeled by differential equations relating the contents (quantity, temperature, pressure) to the flows in and out,
2. Pipes--modeled by equations specifying pressure drops,
3. Junctions--modeled by equations governing mixing and any reactions that occur,
4. Pumps--modeled by equations giving pressure changes in terms of flow, temperature, etc.,

and many other elements for heat exchanges, reaction vessels, etc.

The connections would transmit quantities such as fluid flow rates, temperatures, pressures, and concentrations.

Electrical Networks

Useful types of elements include batteries, current sources, resistors, capacitors, inductors, diodes, and transistors--all modeled by differential or nondifferential equations. Connections transmit voltages and current flows.

2. FORMULATION OF MODEL ELEMENTS

The basic building blocks in the MODEL language are called _elements_. They may be combined to create new elements of greater complexity or they may be used individually as self-contained units. When the definition of an element involves the use of other elements, the component elements are called _subelements_. Elements which employ no subelements are called _primitive elements_. They are defined entirely by equations.

This chapter introduces the MODEL language statements that are used to define MODEL elements. Chapter 3 discusses the use of subelements to create network models. Before a subsystem can be used as a subelement, however, it must first be modeled as an element.

2.1. MODEL Language Structure

In the MODEL language a model is defined by a sequence of text statements called the _input file_. The input file can be punched on cards or entered from some other medium, such as a disk file. MODEL statements can appear anywhere within a 72 character line. There are no fixed character positions at which statements or items within statements must begin. Each input line is a separate statement and therefore the maximum length of any statement is 72 characters. Spaces are significant only as separators between various items which comprise a statement. Any line that begins with a % character is interpreted as a comment and does not affect the model definition.

The MODEL language includes a variety of statement types, each having its own format. The statement type is identified by a keyword, which must be the first nonblank item of the statement. The keyword is followed by specific items for the particular statement type. Examples of MODEL statements are

```
ELEMENT DRAIN
EQUATIONS FLOW(0) + FLOW(1) = 0
```

(Indices in MODEL equations refer to the values of variables at terminals of elements, as explained in chapter 3. In the equation above, FLOW is a terminal variable and the indices refer to the flows at the input and output terminals of the drain element.)

The statements of the input file are grouped by elements. All statements which contribute to the definition of a particular element must appear together in sequence. The first statement of each element definition is an ELEMENT statement which gives the name of the element to be defined, e.g., ELEMENT DRAIN. All statements between this statement and the next ELEMENT statement or the end of the element definitions contribute to the definition of element DRAIN.

The statements within an element definition may appear in virtually any order (certain restrictions for network elements are discussed in section 3.5). Each statement type may be used any number of times. When statements of the same type are grouped together, it is not necessary to include the keyword for each statement. Any statement which does not begin with a keyword is assumed to be of the same type as the last previous keyword, e.g.,

```
EQUATIONS   FLOW(0) + FLOW(1) = 0
            FLOW(0) = SQRT(2*G*A*A*(B(0) - H))
```

The only exception to this rule occurs for statements following an ELEMENT statement. When the statement following an ELEMENT statement does not begin with a keyword, it is assumed to be an EQUATIONS statement. Thus, the first three statements of the definition of element DRAIN could be

```
ELEMENT DRAIN
        FLOW(0) + FLOW(1) = 0
        FLOW(0) = SQRT(2*G*A*A(B(0) - H))
```

The various elements of a model may be defined in any order in the input file. Thus it is necessary for the user to specify the name of the highest level element of the model so the model analysis can begin with that element. This is done with an ANALYZE statement (section 2.3), which is one of several statements which apply to the entire model (i.e., to all elements) and consequently must precede the first element definition of the input file.

In the remainder of this chapter the format of each MODEL statement is presented and the effect of each statement is described in detail. A concise notation is used to present the statement formats. Keywords and delimiters appear in normal type and must appear in the same fashion in specific statements. Underlined items must be supplied by the user. Brackets [] enclose optional items. An asterisk * following a right bracket indicates an optional item which may be repeated any number of times. Vertical bars ¦ inside brackets separate alternatives from which the user may choose one. A space (blank) in this notation indicates that one or more spaces are required at that point in specific statements. Spaces are allowed, but not required, between items which appear in a list and are separated by semicolons. Spaces are also allowed, but not required, between items (e.g., names, operators) in equations.

In the above notation, the format of the EQUATIONS statement, which may include any number of equations separated by semicolons, is

EQUATIONS [<u>lhs</u>=<u>rhs</u>;]*

It is not necessary to include a semicolon after the final equation of the statement. A final semicolon is never needed in any sequence of items separated by semicolons.

2.2. <u>MODEL Elements</u>

Each use of an element in a model is called an <u>instance</u> of that element. Each element need only be defined once, regardless of the number of instances of that element which are used. There are no fixed limits on the number of instances or the number of levels of subelements which can be used in a model. In fact, the only fixed limit on the complexity of a model is the total amount of computer central memory available for the model analysis. There are no fixed limits on the number of instances, variables, equations, connections, etc., which can be used.

The equations for an element are the same in each instance, except that certain variables in the equations, called parameters, may be changed each time the element is used. Once defined, an element may be easily connected to other elements to define new elements. The highest level element is the simulation model itself, which is controlled by run parameters that are assigned each time the model is invoked.

Figure 2.1 illustrates the concepts of elements and subelements in a model of a storm sewer. Element SEWER employs subelements RAIN, RESERV,

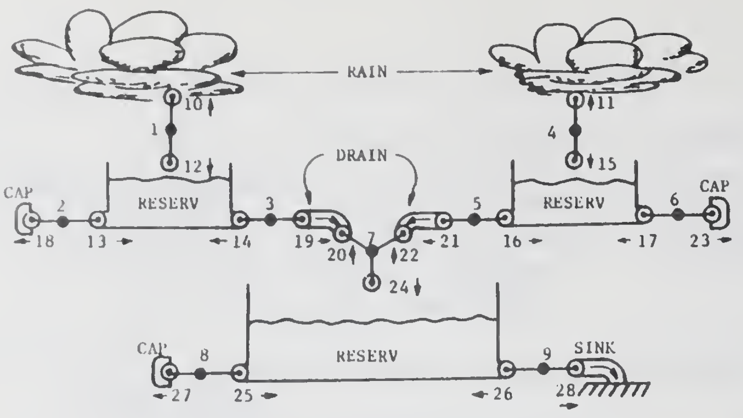


Figure 2.1. Elements and Sub-elements in a Storm Sewer Model

and DRAIN, which model rainfall, reservoirs, and drains, respectively. These subelements are interconnected as shown by the lines joining them in Figure 2.1. Elements RAIN, RESERV, and DRAIN are primitive elements modeled by equations. Element RAIN defines the rate of rainfall entering the sewer. Element DRAIN relates the rate of flow out a drain to the total energy (potential, kinetic, and intrinsic) of the water at the drain. Element RESERV relates the amount of water in a reservoir to the rates of flow into and out of it. The connections transmit the total energy and the rate of flow of the water.

This storm sewer model is discussed further in subsequent sections of this manual and is formulated in the MODEL language in Appendix B.

2.3. ANALYZE Statement

Every model will include one and only one element which is not a subelement of any other. This element is called the highest level element of the model. Since the breakdown of a system into subsystems and then into simpler subsystems must stop at some point, every model will also include one or more primitive elements. The simplest model structure possible consists of one primitive element which is simultaneously the highest level element.

Since the elements may be defined in any order, it is necessary to specify the name of the highest level element. This is accomplished by the ANALYZE statement, which has the form

ANALYZE _name_

where _name_ is the element name of the highest level element. The ANALYZE statement must precede the first element definition. If no ANALYZE statement is given, the highest level element of the model must be the first element defined in the input file.

2.4. Element and Parameter Definition

Elements and their parameters are defined by the ELEMENT statement. Parameters are variables which are specific to an instance of one element, but whose values can be assigned by an element at the next higher level (i.e., an element which uses it as a subelement). The use of parameters minimizes the number of element definitions needed because a parameter can be assigned a different value for each instance of an element. For example, an element which models a pipe might include a parameter which specifies its cross-sectional area. Then instances of the same pipe element could be used to model many pipes, each having a different cross-sectional area. In this case, the value of the parameter would be specified as a constant, but in general, a parameter value can be specified as a variable. For example, a resistance parameter of a resistor element could be specified as a function of time.

At the time an element is defined, parameter values can be assigned for each of its subelements. This is called non-default parameter assignment. Another means of assigning parameter values is default parameter assignment. The definition of an element may include default assignments for any of its parameters. These assignments are used only if the element at the next higher level does not assign values for the parameter. Non-default parameter assignments always take precedence over default parameter assignments.

The name, parameters, and default parameter assignments of an element may all be specified in the ELEMENT statement, which has the general form

```
ELEMENT name [parm[(d1[,d2])][=defexp];]*
```

name is the name of the element. All element names used in MODEL must be alphanumeric strings of eight characters or less and must begin with a letter. parm represents a parameter name, an alphanumeric string of seven characters or less beginning with a letter. MODEL parameters may be scalars or arrays of one or two dimensions. The dimensions of parameter arrays are defined in the form parm (d1) or parm (d1,d2), where d1 and d2 represent the integer dimensions. The dimensions may be omitted for scalar parameters.

Default parameter values are assigned in the form parm = defexp or parm (d1) = defexp or parm (d1,d2) = defexp, where defexp represents any well-defined model expression whose variables are known in element name and whose dimensions are the same as those of parm. The allowable forms for scalar and array expressions are described in section 2.9. Default parameter assignments may be omitted when not desired.

The drain element used in the storm sewer model of Figure 2.1 has two scalar parameters: A, the cross-sectional area of the open drain, and H, the height of the opening (above some arbitrary reference point). If no default parameter values are assigned, the ELEMENT statement for this element is

```
ELEMENT DRAIN A ; H
```

To assign a default area of 1 and a default height of 0 for the drain, the element statement would be

```
ELEMENT DRAIN A = 1 ; H = 0
```

There is no fixed limit on the number of parameters an element can have. Many parameter definitions can be included in the ELEMENT statement. If an ELEMENT statement will not fit on one input line, the DEFAULTS statement can be used to define additional parameters. This statement has the general form

DEFAULTS [<u>parm</u>[(<u>d1</u>[,<u>d2</u>])][=<u>defexp</u>];]*

where the meaning of each item is as described for the ELEMENT statement. Any number of DEFAULTS statements may be used in any element.

Non-default parameter values are assigned by the SUB-ELEMENT and NON-DEFAULTS statements when the element is used as a subelement. These are described in section 3.5. However, parameters of the highest level element of a model require a different kind of non-default assignment. These parameters are called <u>run parameters</u> because their values are assigned in the <u>run control program</u> which invokes the numerical simulation of the model. The use of run parameters allows a simulation to change the values of these parameters. Run parameters are discussed in section 4.3.

2.5 MODEL Variables

All element behavior is modeled by real-valued variables, whose relationships are specified by equations and connections. Five types of variables can be used. These are parameters, local variables, global variables, terminal variables, which are transmitted through connections, and a single independent variable, which usually represents time.

MODEL variable names can be any string of seven or less alphanumeric characters beginning with a letter. All dependent variables may be scalars or arrays of one or two dimensions. Parameters are discussed in connection with element definitions in section 2.4. Local and global variables and the independent variable are described below in sections 2.6 through 2.8. These variables are most commonly used in the equations of primitive elements. Terminal variables represent quantities that are passed through connections when an element is used as a subelement. They are discussed in conjunction with network models in chapter 3.

2.6. Local Variables

Some elements may use variables that are specific, or local, to each instance. For example, the length of a structural beam may depend on both the applied stress and its internal temperature. That temperature is a function of the amount of energy dissipated in the beam, and of the loss of heat to the environment. Such variables are defined as local variables. Each instance of the beam will then have its own temperature variable, whose behavior is defined by a differential equation.

Like parameters, local variables may be scalars or arrays, and no fixed limit is imposed on the number of local variables an element can have. Local variables may only be referenced in the equations of the element in which they are defined.

Local variables are defined by the LOCALS statement, which has the general form

LOCALS [var[(d1[,d2])];]*

var represents a local variable name and the optional integer dimensions are represented by d1 and d2. Any number of LOCALS statements may be used in any element.

The reservoir element used in the storm sewer model of Figure 2.1 has five scalar local variables; HS, V1, P1, V2 and P2, which represent the surface height of the reservoir and the fluid velocities and pressures at its two outlets. (One of the two outlets of each instance of the reservoir element is capped, so there is only one drain attached to each reservoir.) The LOCALS statement for this element is

LOCALS HS ; V1 ; P1 ; V2 ; P2

2.7. Global Variables

Global variables are variables which are common to all instances throughout an entire system or subsystem model. Typically these are environmental variables such as the ambient temperature of an electrical network. They are variables because they may be influenced by the system behavior. For example, the ambient temperature of an electrical network in a closed box depends on the heat dissipated by the network inside that box.

Global variables allow information to be passed between instances independently of the way in which the instances are interconnected. A variable which is defined as global in one element is automatically defined in all subelements of that element, in all of their subelements, etc., and may be referenced in the equations of any of these elements. The global variable name need not, and should not, be redefined in any of the subelements.

Like parameters and local variables, global variables may be scalars or arrays, and no fixed limit is placed on the number of global variables an element can have.

Global variables are defined by the GLOBALS statement, which has the general form

GLOBALS [var[(d1[,d2])];]*

var represents a global variable name and the optional dimensions are represented by d1 and d2. Any number of GLOBALS statements can be used in any element.

The storm sewer model of Figure 2.1 has two scalar global variables; GAMMA and G, which represent the density of the fluid (water) and the acceleration due to gravity. Although these will be specified as constants, they are represented by global variables for clarity and to allow their values to be changed easily throughout the entire model if necessary. The GLOBALS statement for element SEWER is

GLOBALS GAMMA; G

2.8. Derivatives and the Independent Variable

The MODEL language includes a single independent variable which can be referenced in the equations of any element. The independent variable is usually named TIME. However, it can be renamed by the RENAME statement, which has the general form

RENAME name

where name represents the name by which the independent variable is to be known. The RENAME statement must precede the first element definition of the input file.

All derivatives referenced in MODEL equations are taken with respect to the single independent variable. Derivatives are specified by an apostrophe after the variable name, e.g., HS'. The derivative operator may be applied to scalar or array variables. Higher order derivatives are specified through the introduction of auxiliary variables. For example, the introduction of the variable DX defined by the equation

DX = X'

allows the second derivative of X to be referenced as DX'. The independent variable is always a global variable and does not need to be defined explicitly.

2.9. Equations

MODEL offers a very flexible, convenient and natural format for ordinary differential and nondifferential equations. Equations may be formulated implicitly. This means that the general form of an equation is simply

<u>lhs</u> = <u>rhs</u>

where <u>lhs</u> and <u>rhs</u> represent arbitrary well-defined expressions involving constants, parameters and variables, and built-in and user-defined operators and functions. (Parameter assignments are a special case of this form in which the left hand side is a single parameter.) There is no need for the user to derive an explicit formulation of the equations. Equations may be algebraically manipulated in any fashion without changing their interpretation. The differentiation operator (') may be applied to any variable in any expression. Thus any first order ordinary differential or nondifferential equation which expresses a desired relationship between variables can be entered directly.

The equations in (1.7) for the motion of a projectile could be formulated in an element as shown below. The gravitational acceleration and the initial velocity of the projectile have been defined as parameters of the element.

```
ELEMENT PROJECT G;V
X' = S
S' = -G
LOCALS S; X
```

The initial values for S and X are specified using the INITIAL statement described in section 2.11. The complete formulation of this problem is shown in appendix B.

MODEL variables and parameters may be scalars or they may be arrays of one or two dimensions. Array expressions and equations may be entered directly. To facilitate this, the common arithmetic operators for addition (+), subtraction (-) and multiplication (*) may be applied to arrays as well as to scalars and an operator is included for vector product (.). FOBTRAN functions supplied by the user can also be applied to arrays, as can the differentiation operator. Array expressions involving array variables and operators may appear in any equation. For example, if F is a 4 x 4 array of coefficients and X is a 4 x 1 vector of state variables, the single array equation

X' = F * X

can be used in place of the four associated scalar equations. MODEL does not support array indexing in equations; equations must operate on arrays as indivisible entities. However, indexing is possible at the FORTRAN level in user-supplied functions, as explained in section 2.15.

Equations are specified by the EQUATIONS statement, which has the general form

EQUATIONS [lhs=rhs;]*

(If the EQUATIONS statement is the first statement after an ELEMENT statement, it is not necessary to include the keyword EQUATIONS.) The equations of an element may reference any global, local or parameter variables of the element and any terminal variables associated with terminals of the element (except I-set variables at local terminals) in a network model. The built-in operators which can be used in equations are listed in Table 2.1. With the exception of the division (/) and exponentiation (**) operators, all of these may be applied to arrays as well as to scalars. The built-in functions are listed in Table 2.2.

In addition to providing a standard set of built-in arithmetic operators, MODEL allows the user to interface his own FORTRAN functions and subroutines to the MODEL package and to utilize these user-defined functions in the equations of his model. This allows virtually any

Table 2.1. Built-in Operators.

'	differentiation	(scalar or array)
**	exponentiation	(scalar only)
/	division	(scalar only)
*	multiplication	(scalar or array)
.	vector product	(one-dimensional arrays only)
+	addition	(scalar or array)
-	subtraction	(scalar or array)
()	parentheses change the order of expression evaluation	

Table 2.2. Built-in Functions

x's represent scalar arguments

LOG(x)	natural logarithm of x
EXP(x)	e raised to x power
SQRT(x)	square root of x
ABS(x)	absolute value of x
ATAN(x)	arc tangent (in radians) of x
COS(x)	cosine of x (in radians)
SIN(x)	sine of x (in radians)
MAX(x1,x2,...,xk)	maximum of from 1-10 arguments
MIN(x1,x2,...,xk)	minimum of from 1-10 arguments

function or operation not provided by MODEL to be implemented in FORTRAN, as described in Section 2.15.

With the exception of the differentiation operator, which may only be applied to variables, all of these operators may be applied to variables or to expressions containing any other operators. Expressions are evaluated in the usual order (the same as FORTRAN) and parentheses may be inserted to change the order of evaluation.

The order in which the equations appear within an element is of no consequence because all equations will be satisfied (within error tolerance) simultaneously during the numerical simulation.

All variables and parameters in a model are represented as real-valued quantities during the numerical simulation. The precision (either single or double) to which these quantities are computed depends on the word length used in the host computer and will be selected by MODEL. Constants may be entered as either integers or real numbers, e.g., 3, 3.0, 3.0E0, and 3.0+D0. However, in the case of integer exponents the user is urged to write X**3 rather than X**3. or X**3.E0, etc., and constants used as arguments in function and subroutine references must be consistent with the argument types in the subprogram definition. All built-in functions require real-valued arguments.

2.10. Conditional Equations

An EQUATIONS statement can be structured via IF, THEN, and ELSE clauses to create state dependent elements whose defining equations are altered when certain thresholds in the continuous variables are attained. The general form of a conditional equation is

```
EQUATIONS IF cond THEN lhs=rhs
          [ELSE IF cond THEN lhs=rhs]*
           ELSE lhs=rhs
```

Each use of lhs=rhs represents a different equation. Each use of cond represents a relation or condition on the variables and parameters of the element which must hold for the associated equation to take effect. The relational operators < (less than) and > (greater than) can be used to define threshold relations between arbitrary scalar expressions (e.g., X < Y + Z). The logical operators & (and) and ¦ (or) can be used to define conditions consisting of logically combined threshold relations (e.g., X < Y + Z & A > B).

Conditional expressions are evaluated according to FORTRAN operator precedences. This means that the condition

```
A < B ¦ C > D & E > F
```

is evaluated as

```
(A < B) ¦ ((C > D) & (E > F))
```

and the condition

```
A < B & C > D ¦ E > F
```

is evaluated as

```
((A < B) & C > D)) ¦ (E > F)
```

The user is encouraged to insert parentheses in conditional expressions to be certain of their interpretation.

The keyword EQUATIONS may be omitted if the conditional equation follows immediately after an ELEMENT statement or another EQUATIONS statement. Two or more alternative equations may appear on the same line if they are separated by semicolons, e.g.,

IF <u>cond</u> THEN <u>lhs</u>=<u>rhs</u> ; ELSE <u>lhs</u>=<u>rhs</u>

The use of conditional equations is illustrated by the following example, which is an extension of the spring element presented in section 3.8. The conditional equations describe the nonlinear operation of a spring when a threshold length is reached or exceeded. The spring's undeformed length, L, is a local variable which changes when the spring is stretched past the threshold of DELMAX. The conditional equation causes L to increase according to the modulus of stretching, C, when DELMAX is exceeded and the length is increasing.

```
        F(1) + F(2) = 0
        K*(X(2)-X(1)-L) = F(2)
        IF X(2)-X(1) < DELMAX+L ¦ X(2)'-X(1)' < 0 THEN L' = 0
        ELSE L' = C*(X(2)' - X(1)')
```

X(0) and X(1) represent the positions of the endpoints of the spring and F(0) and F(1) represent the forces exerted on these endpoints. The condition on the equation for L allows deformation to occur when the threshold is exceeded and the length is increasing. Otherwise, the length is fixed by the L' = 0 equation. The deformation equation in the ELSE clause is obtained by differentiating the usual deformation equation to obtain an equation in terms of L'. Any discontinuity that is introduced by the switching of the conditional equations will therefore occur in the derivatives of the solution variables. The solution itself will remain continuous, a necessary condition for a well defined problem.

If the conditional equations are formulated in terms of the solution variables, then both alternative equations must be satisfied by the solution at the point of transition. This restriction is necessary to guarantee that the solution will be continuous at the point of transition.

A final consideration in the use of conditional equations is the specification of the initial state. To make certain that the correct steady-state solution is obtained, all conditional expressions should be well defined in terms of the initial values that are supplied for the problem. For the spring element, the initial locations of the ends of the spring, X(1) and X(2), and the initial length of the spring, L, should be specified to force linear operation of the spring element. This prevents changes in the conditional equations during the course of determining the steady-state solution. In general, any such changes introduce discontinuities into the steady-state equations and should be avoided. Initial values are assigned by the INITIAL statement, described in section 2.11.

2.11. Initial Values

Most models will require the specification of initial values for some variables in order to define the initial state of the system. For example, initial values would be needed for the undeformed length of the spring element of the previous section and for the surface height of the reservoir element of section 2.2. Initial values for variables of an element are specified by the INITIAL statement which has the general form

INITIAL [_var_ = _const_;]*

var represents a variable or derivative of the element. _const_ represents any expression involving only constants, run parameters, and parameters or variables which have been equivalenced to one of these. _var_ and _const_ may be scalars or arrays. _const_ may depend on parameters of the element. This allows different initial values to be used for each instance. For example, the definition of the reservoir element in appendix B includes a parameter HINIT and the statement

INITIAL HS = HINIT

This allows the initial surface height to be specified independently for each reservoir used in the storm sewer. Parameterization of initial values in the highest level element of a model allows these initial values to be specified in the run-control program. This enables the simulation to be repeated a number of times without repeating the model analysis process to change these initial values. The projectile element of section 2.9 would use the following statement to initialize S and X:

```
INITIAL S = V ; X = 0
```

The initial value for S is a run parameter which may be changed in the run control program.

The initial value of the independent variable is specified through the TSTART control variable. The value assigned to TSTART is the value of the independent variable when the simulation begins. The default value for TSTART is 0. A non-default value for TSTART can be specified in the run control program (Chapter 4).

At the beginning of a simulation the independent variable assumes the value of TSTART and the initial values specified by the user are assigned. MODEL then computes initial values for the remaining variables and derivatives. Thus, it is not necessary for the user to specify initial values for every variable and derivative of the model. However, to specify the initial state of the model uniquely the user must specify one initial value for each differentiated variable which appears (e.g., S' and X' in the projectile element above). If an initial value is not specified for some differentiated variable, the model will not have a unique initial state. In these cases MODEL will compute an initial state in which the initial value of the variable is 0, if such a state exists. Variables whose derivatives do not appear in the model need not be initialized by the user in order to obtain a unique initial state. However, specification of initial values for these variables simplifies the computation of the initial state. Therefore, the user is encouraged to specify all initial values which are known.

2.12. Termination Conditions

Every model must include at least one termination condition to specify the condition or conditions under which the simulation is to be terminated. Termination conditions can be specified in the equations of any element by the conditional STOP statement. The general form of this statement is

```
EQUATIONS IF cond THEN STOP[=n]
```

where _cond_ represents any condition on the variables and parameters of the element as described in section 2.10 and _n_ represents an optional termination code which is an integer constant. An ELSE clause is not allowed after a conditional STOP statement.

A termination condition might be a simple threshold on the independent variable, e.g.,

```
IF TIME > 100 THEN STOP
```

or it might be a condition on variables and parameters of an element which represents either normal or abnormal termination. For example, the statement

```
IF I(1) > IMAX THEN STOP = 1
```

might be included in an element which models a fuse to terminate the simulation if the fuse blows. If this condition arises during the simulation, the simulation will stop, the termination code 1 will be printed and assigned to the control variable ISTOP, and control will return to the run control program. The printed termination code communicates the termination condition to the user while the control variable ISTOP allows the run control program to alter the control sequence according to the circumstances which caused termination. The use of control variables in the run control program is discussed in section 4.2.

If a conditional STOP statement does not specify a termination code, MODEL will use the instance number of the element containing that STOP statement as its termination code. A unique instance number is assigned to each element instance in a model and these instance numbers are listed in the model dictionary which is discussed in section 5.2.

The use of instance numbers as termination codes allows a different termination code to be used for each instance. This allows the user to determine the particular element which caused the simulation to terminate. Parameterization of termination conditions allows these conditions to be controlled by the invoking element. In this way fuses of different amperage ratings can be modeled by a single element. Parameterization of termination conditions in the highest level element of a model allows these conditions to be specified through the run control program. This allows the simulation to be repeated a number of times without repeating the model analysis process to change the termination conditions.

A simple termination threshold on the independent variable can also be specified in the run control program through the control variable TSTOP. For example, the assignment

```
TSTOP = 10.
```

in the run control program causes the simulation to be terminated when the value of the independent variable exceeds 10. When the simulation is terminated in this fashion the termination code will be zero.

TSTOP refers to the units in which the independent variable is modeled. If the independent variable is time, these units might be seconds, minutes, hours, days, etc. For example, time is modeled in seconds in the storm sewer model of appendix B. The desired duration of the simulation is two hours so the run-control program for this model includes the assignment

```
TSTOP = 7200.D0
```

The default value for TSTOP is 1. The user must specify a non-default value for TSTOP in the run control program if he wants the simulation to continue beyond a value of 1 for the independent variable, even if he is relying on termination conditions specified within the model to terminate the simulation.

2.13. Output Variables

Every model will require the specification of some output variables whose values are to be recorded for output during the simulation. Output variables may be specified in any element by the OUTPUT statement. This statement has the general form

OUTPUT [<u>var</u>;]*

where <u>var</u> represents the name of a variable or parameter of the element. For example, in the storm sewer model of appendix B the desired output variables are the surface heights of the three reservoirs. Thus the reservoir element includes the statement

```
OUTPUT  HS
```

To record a derivative, a dummy variable must be introduced and set equal to the derivative in an equation. The dummy variable can then be used in an OUTPUT statement.

The values of the output variables are recorded at regular intervals during the simulation. The communication interval is specified through the control variable COMINT. The interval in the independent variable between recording operations is simply the value assigned to COMINT. The default value for COMINT is .01. A non-default value for COMINT can be assigned in the run control program. If the value zero is assigned to COMINT, the values of the output variables are recorded at every step of the simulation.

The value of the independent variable automatically becomes the first value recorded at each communication interval. Thus, there is no need for the user to specify the independent variable as an output variable.

The set of output variable values recorded during the simulation is called the solution history. The solution history can be recorded on any two logical unit numbers between 1 and 88 excluding unit 5. Unit 6, the standard line printer unit, is the default unit for printed output. A second logical unit number can be specified to save the solution for additional processing. Unit 1 is the default unit for this purpose. The logical unit numbers are set by control variables in the run control program as explained in section 4.2. If the default unit numbers are changed, the user must provide appropriate job control for the numerical simulation to relate the selected logical unit to the desired output device. This is discussed in appendix C.

A title for the solution history can be specified by the TITLE statement, which has the general form

TITLE _tstring_

where _tstring_ represents any character string. The TITLE statement must precede the first element definition of the input file.

The user is free to examine and alter the FORTRAN subroutines which record the output so great flexibility in data recording is available at the FORTRAN level. The FORTRAN subroutine RECORD, which records the solution history, is shown for the models in appendix B. RECORD is called twice at the beginning of the simulation--first to record the solution history header which consists of the title (if any) and a set of labels which are the names of the output variables, and then to record the initial values of the output variables. Thereafter, RECORD is called at the end of every communication interval to record the updated values of the output variables.

RECORD has four arguments; INIT, IUNIT, N and A. INIT is an integer flag which is zero when RECORD is called to record the solution history header and is one thereafter. IUNIT assumes the logical unit numbers specified in the model or the default unit numbers six and one. The values of the output variables are passed to RECORD in the vector A, whose length is N. The labels in the solution history header indicate the order in which the output variables appear in A. In cases of two or more instances of an element which has an output variable (e.g., variable HS in the reservoir instances of the storm sewer model of appendix B), these output variables appear in increasing order of their associated instance numbers (see section 5.2). Thus in subroutine RECORD of appendix B, the first label HS refers to the surface height of the upper left reservoir which is instance 4, the second label HS refers to the surface height of the upper right reservoir which is instance 5, and the third label HS refers to the surface height of the lower reservoir which is instance 10. After determining the order of appearance of the output variables in the vector A, the user familiar with FORTRAN should find it quite easy to alter RECORD to perform more sophisticated output operations. For example, calls to plotting subroutines could be added to produce graphical output from the simulation.

Subroutine RECORD is called by subroutine REPORT. The sole purpose of REPORT is to extract all variables which are designated as output variables and to assign them to the A array before calling RECORD. If it is desired to add or delete output variables, this may be accomplished in REPORT without repeating the model analysis. The FORTRAN names of the MODEL variables may be obtained from the model dictionary discussed in section 5.2. The dimension of the A array may also be changed in REPORT according to the output requirements.

The user may obtain a listing of subroutines RECORD and REPORT by specifying either the FLIST or the SLIST option for the model. The FLIST option causes all FORTRAN code produced by the model analysis process to be listed while the SLIST option causes only the run control program and the output subroutines to be listed. The FLIST option is specified by the statement

```
FLIST
```

and the SLIST option is specified by the statement

```
SLIST
```

These statements must precede the first element definition of the input file.

In order to change subroutines RECORD and REPORT, the user should use the listing produced by the model analysis process as a starting point, make the desired changes, and substitute the new versions of the subroutines for the old ones in the set of subroutines used by the numerical simulation process. This requires that the model analysis process and the numerical simulation process be run as separate jobs to allow the user to intervene between them. The manner in which this can be done is discussed in section 4.6, which describes how the user can alter the FORTRAN code produced by the model analysis process before invoking the numerical simulation process.

2.14. Table Functions

Empirical data functions of one variable can be defined in tabular fashion in the MODEL language. This is done by the TABLE statement, which follows the last element definition of the input file (see appendix A). The TABLE statement has the general form

```
TABLE  tnum
 t1    x1
 t2    x2
 .      .
 .      .
 .      .
 tn    xn
```

tnum reprsents a unique positive integer, called the table number, by which the specific table function is identified. The _ti_ represent the points at which the function values, _xi_, are known empirically. The _ti_ must appear in increasing order. Any number of table functions may be defined. They may be referenced in expressions of any element in the form TABLE (<u>tnum</u>,<u>arg</u>) where <u>tnum</u> is the table number and <u>arg</u> is an expression specifying a t value at which the table function is to be interpolated.

2.15. User-Defined Functions

The user can define his own FORTRAN function and subroutine subprograms to evaluate special functions needed in equations, conditions, and parameter and initial value assignments. Scalar functions can be defined as function subprograms while array functions--that is, those which return array results--are defined as subroutines. (FORTRAN does not support function subprograms which return array results.) Both types of subprograms are referenced as functions in MODEL expressions. When a subroutine is defined, its array result must be the last dummy argument of the definition. When the subroutine is referenced in a MODEL expression, this argument is omitted as is the CALL statement. For example, a subroutine which computes the transpose of an array might be referenced in FORTRAN by the CALL statement

```
CALL TRANSP(N,M,A,AT)
```

where A is an M x N array whose N x M transpose, AT, is to be computed. This subroutine could be invoked in a MODEL expression by the function-like reference TRANSP(N,M,A), e.g.,

```
EQUATIONS B = A*C*TRANSP(4,4,A)
```

MODEL will automatically generate the necessary FORTRAN CALL statement and insert the result into the expression in place of the reference TRANSP(4,4,A). This permits the convenience of function-like referencing for both scalar and array functions. There is no need for the user to define an auxiliary variable AT and call the subroutine explicitly in the form

```
CALL TRANSP(4,4,A,AT)
B = A*C*AT
```

This enables the user to save a significant amount of effort when several such subroutine references are required and allows expressions containing array functions to be written in a more natural fashion.

Each FORTRAN subprogram referenced in a model must be defined by a FORTRAN source or object subprogram and its syntax (that is, the number of arguments it needs and the dimensions of its result) must be defined by a FUNCTION or SUBROUTINE statement in the model definition.

The FORTRAN subprogram performs the function evaluation during the numerical simulation. It may be defined by passing the FORTRAN source code through the model analysis process, by including the source code in the FORTRAN step of the numerical simulation process, or by compiling the source code separately and including the object code in the link-load step of the numerical simulation process. If the FORTRAN source code is passed through MAP, it must be put in the run control file, as explained in section 4.4. The manner in which a FORTRAN subprogram can be interfaced at the FORTRAN or link-load step of NSP is discussed in section 4.5.

The FUNCTION and SUBROUTINE statements enable MODEL to recognize subprogram references in expressions, to check these references for correctness in the number of arguments and the dimensions of the result, and, in the case of subroutines, to insert the array result into the expression which references the subroutine. The subprogram syntax statements have the general forms

```
        FUNCTION name n
        SUBROUTINE name n [RVECTOR [d2]|
                           CVECTOR [d1]|
                           ARRAY [d1 d2]]
```

where name represents the name of the function or subroutine and n represents the number of arguments used in referencing the subprogram in MODEL expressions. For a subroutine which is defined with M dummy arguments (the last of which is its result), n should be M-1 because the last argument is omitted when the subroutine is referenced in MODEL expressions. For example, the subprogram syntax statement for the subroutine TRANSP mentioned above would be

SUBROUTINE TRANSP 3

The FUNCTION statement specifies the syntax of references to a FORTRAN function subprogram, which by definition must return a scalar result. The SUBROUTINE statement specifies the syntax of references to a subroutine subprogram which returns an array result. The RVECTOR, CVECTOR or ARRAY clause of the SUBROUTINE statement specifies whether the result is a row vector (that is, 1 x d2), a column vector (d1 x 1), or an array (d1 x d2). If the dimensions of the result are fixed, they are specified by the integer(s) d1 and/or d2. If the dimensions of the result depend on the actual arguments used in referencing the subroutine, no dimensions are specified in the SUBROUTINE statement. In this case, the dimensions of the result must be the first argument (for vectors) or the first two arguments (for arrays) of the subroutine and only integer constants may replace these dummy arguments when the subroutine is referenced in expressions. This is the manner in which the subroutine TRANSP mentioned above would be referenced. If the SUBROUTINE statement includes no RVECTOR, CVECTOR, or ARRAY clause, the result is assumed to be an array whose dimensions are specified in this way.

The FUNCTION and SUBROUTINE statements must precede the first element definition of the input file and no more than 24 subprograms may be referenced in the model. Any well-defined FORTRAN subprograms are allowed and these may be referenced in any element.

3. NETWORK MODELS

All network models contain two types of components:

1. A set of basic activities such as the behavior of a diode or a single chemical reaction. These are called elements and are the subject of chapter 2.
2. Interactions between the basic elements such as current flow from a resistor to a diode or the influence of the concentration of a radical on a chemical reaction. These are called connections and they provide the mechanism for creating network models.

The MODEL language can accommodate a wide variety of types of connections, including electrical and fluid junctions, structural and mechanical joints, and many others. The connection points of an element are called terminals (e.g., the pins of a transistor, the ends of a beam, or the input to a chemical reaction element), and with each terminal are associated certain terminal variables (e.g., electrical voltage and current, components of displacement and force, or radical concentration and rate of flow).

Terminals are connected into nodes (a node is simply a set of connected terminals) to indicate natural relationships between their associated terminal variables, which are automatically transmitted through the connections. Thus, three resistors might be connected into a node to indicate that the voltage applied to those resistors is the same and the directional sum of the currents drawn by the resistors is zero.

3.1. Terminal Variable Sets and Node Relationships

Terminal variables obey conservation laws at connections. These represent generalizations of Kirchoff's voltage and current laws for electrical networks and hence the terminal variables associated with each terminal of a given type are divided into two sets, called the E-set and the I-set.

E-type terminal variables (the members of E-sets) represent intrinsic quantities such as voltage or position. Each member of an E-set automatically assumes the same value at every terminal connected to a common node. This node relationship is called Kirchoff's voltage law for electrical networks. It describes many naturally occurring relationships such as the relationship between voltages on connected lines in electrical networks, the relationship between the various components of displacement at connected members in structural networks (if the ends of several members are joined, their future displacements will be identical), and the relationship between the total energy of a fluid at various points along a common flow line (in a steady flow situation) in hydraulic networks.

I-type terminal variables (the members of I-sets) represent directional quantities such as current, force, or fluid flow (mass/unit time) which are conserved over a node. By convention, I-type terminal variables are positive when directed into an element at its external terminals (external terminals are those which are connected to other elements) and they are negative when directed out of an element at its external terminals. Likewise, they are positive when directed out of an element at the external terminals of its subelements (because in such cases they are directed into the subelements) and they are negative when directed into an element at the external terminals of its subelements. Under this sign convention, each member of an I-set automatically sums to zero over all external terminals connected to a common node. This ensures that the quantities represented by I-type terminal variables do not accumulate in any node. For example, if two instances of a pipe element are invoked as subelements and the pipes are joined end-to-end to the same local terminal in the invoking element, as shown in Figure 3.1 below,

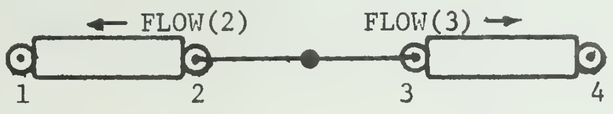


Figure 3.1. Flow Variable Sign Conventions in a Pipe Element.

the flow variable at the right-most external terminal of the left pipe is positive if and only if fluid is flowing from right to left and the flow variable at the left-most external terminal of the right pipe is positive if and only if fluid is flowing from left to right. Since all fluid flowing out of one pipe flows into the other, the two flow variables sum to zero.

This node relationship is called Kirchoff's current law for electrical networks, Newton's second law for structural networks, and the conservation of mass for hydraulic networks. It describes many naturally occurring relationships such as the relationship between connected current branches in electrical networks, the relationship between the various components of force at connected members in structural networks, and the relationship between connected flow branches in hydraulic networks.

If a desired terminal variable is not of E-type or I-type, it can often be transformed into an I-type variable. Since the zero sum rule at nodes is a form of a conservation equation, choosing variables that are conserved (e.g., momentum, energy, mass, current) will usually give an I-type variable. For example, one might be tempted to use pressure, flow, and temperature to characterize a water pipe connection. However, at a junction of more than two pipes, temperature is not of either terminal variable type. If, instead, one uses heatflow (temperature*flow), this is an I-type variable because heat satisfies an energy conservation law.

The terminal variables associated with the terminals of an element may be referenced in the equations of that element. For example, if a resistor element is defined with external terminals numbered 0 and 1 as shown in Figure 3.2 below,

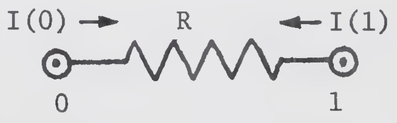


Figure 3.2. Terminal Variables in a Resistor Element.

and if these terminals are of type ELECTRIC with an E-set consisting of V, the voltage, and an I-set consisting of I, the current, then the voltage at terminal 0 would be referenced as V(0) and the current directed into the resistor at terminal 1 would be referenced as I(1).

Current directed out of the resistor at terminal 1 would be referenced as -I(1). During the numerical simulation, I(1) will be positive if and only if current is flowing right to left and I(1) will be negative if and only if current is flowing left to right.

The equations for this element are:

```
I(0) + I(1) = 0
V(1) - V(0) = I(1)*R
```

The first equation states that the current directed into one end is the negative of the current directed into the other end. (Although I-type terminal variables satisfy conservation laws on nodes, the same is not always true in elements. For example, the flows into a storage tank may not sum to zero.) The second equation is Ohm's law.

Note that these equations model the resistor correctly regardless of the direction in which current flows during the simulation. As long as the equations are consistent with the sign convention for I-set members, e.g.,

```
V(1) - V(0) = I(1)*R
```

and not

```
V(1) - V(0) = I(0)*R
```

the equations will hold for current in either direction.

The derivatives of the terminal variables of an element may also be referenced in the equations of that element. For example, in the resistor element above, the derivative of the current directed into the resistor at terminal 0 would be referenced as I(0)' (not as I'(0)).

3.2. Terminal Definitions

Terminals may be either external or local in scope. The external terminals of an element are its points of connection to terminals outside its own subsystem. When the element is invoked as a subelement, each of its external terminals is connected to a terminal (either external or local) in the invoking element. Thus, its external terminals are the means of connecting the element into a network (e.g., the pins of a transistor). The local terminals of an element are points of connection to other terminals in its subsystem. Local terminals are necessary only if the element (e.g., a transistor) is defined as a network of interconnected subelement instances. They provide a flexible means of interconnecting the external terminals of its subelements. Local terminals may also be connected to other local or external terminals of the same element. However, because I-type terminal variables can neither be directed into nor out of an element at its local terminals, local terminals do not enter into the node relationships for I-set members.

Each terminal of an element is distinguished by a unique, non-negative integer called its terminal ID number. Terminal ID numbers provide the means of identifying terminal references in terminal definition and connection statements and the means of identifying terminal variable references in equations.

Terminals are defined with local scope by the TERMINALS statement. Defined terminals are given external scope by the EXTERNALS statement. The basic formats of these statements are the same. Terminals are identified by ID numbers and may be referenced singly, in consecutively numbered sequences, or in any combination of these fashions, as shown in the examples below.

```
TERMINALS  1  2
EXTERNALS  2-5
TERMINALS  3-5  10  6-8
```

EXTERNALS statements may appear either before or after the TERMINALS statements which define the terminals. Terminals need not be defined in the order of their ID numbers and the set of terminal ID numbers defined in an element need not be consecutive. No fixed limit is imposed on the number of local or external terminals which can be defined in an element. Any number of TERMINALS and EXTERNALS statements can be used in any element.

3.3. Terminal Type Assignment

Because physically different terminals may be present in a single model (e.g., electrical control lines, air ducts, and coolant pipes might be present in a model of an air conditioning system), the terminals can be classified into as many different terminal types as necessary. A terminal type, which is indicated by name, may optionally be associated with each terminal or group of terminals defined in a TERMINALS statement. The general form of the TERMINALS statement is

TERMINALS [id1[-id2] [type]]*

The general form of the EXTERNALS statement differs only in the keyword and the absence of the optional item type.

EXTERNALS [id1[-id2]]*

Each use of id represents a terminal ID number and type represents the associated terminal type. For example, if CONTROL is the name of a terminal type which models an electrical control junction and WATER is the name of a terminal type which models a water pipe junction, the statement

```
TERMINALS  1  CONTROL  3  2  WATER
```

defines terminal 1 with type CONTROL, terminal 2 with type WATER, and terminal 3 with no associated terminal type.

It is not necessary to define the type of every terminal explicitly. Connections between terminals of different types are not meaningful and therefore are not allowed. Thus the model analysis process can determine the types of many terminals from the types of terminals to which they are connected. A terminal's type will be undefined only if no terminal to which it is connected, either directly or indirectly, is assigned a type. In practice it suffices to assign types to the terminals of each primitive element used.

Although the user is prevented from connecting terminals of different types to a common node (so one cannot plug a transistor into a water pipe), it is possible to define an element to interface between terminal types. For example, a pressure transducer element could have one external terminal of type CONTROL and one external terminal of type WATER. Its function (accomplished by an equation) would be to set the control level transmitted through the CONTROL terminal equal to the water pressure transmitted through the WATER terminal.

3.4. Terminal Type Definition

The definition of a terminal type consists of the specification of its terminal type name and of the members of its associated E-set and I-set. Terminal types are defined by the TYPE statement, which has the general form

```
     TYPE tname [E([ename[(d1[,d2])];]*)][I([iname[(d1[,d2])];]*)]
```

tname represents the terminal type name. *ename* and *iname* represent names of members of the E-set and the I-set, respectively. The members of the E-set are listed inside the parentheses which follow the designator E, and are separated by semicolons. The members of the I-set are listed similarly following the designator I. Like parameters and global and local variables, terminal variables may be scalars or arrays. The optional dimensions of each terminal variable are specified in the form *name* (*d1*) or *name* (*d1*,*d2*).

TYPE statements must consist of 72 characters or less, but otherwise there is no restriction on the number of terminal variables which can be associated with any terminal type. Either the E-set or the I-set may be omitted if no variables of one type are required. TYPE statements must precede the first element definition of the input file. Once defined, a terminal type may be used in any element of the model. There is no fixed limit on the number of terminal types which can be defined or the number which can be used in any element.

The TYPE statements of Table 3.1 illustrate the terminal type definition format and the use of E-type and I-type terminal variables to model various connective effects. The symbol % denotes a line of comment text in the MODEL language. The E-type and I-type node relationships are

Table 3.1. Terminal Type Definitions

```
% ELECTRICAL JUNCTION
  TYPE ELECTRIC E(VOLTAGE) I(CURRENT)

% CONTROL JUNCTION
  TYPE CONTROL E(LEVEL)

% WATER PIPE JUNCTION
  TYPE WATER E(PRESSURE) I(FLOW ; HTFLOW)

% CHEMICAL REACTION CONNECTION
  TYPE RADICAL E(CONC) I(FLOW)

% STRUCTURAL PIN JOINT:MODELED IN TERMS OF
% FORCES AND DEFLECTIONS IN THREE DIMENSIONS
  TYPE PINJOINT E(DEFLX;DEFLY;DEFLZ) I(FORCEX;FORCEY;FORCEZ)

% STRUCTURAL JOINT:MODELED IN TERMS OF STRESS
% AND STRAIN TENSORS IN TWO DIMENSIONS.
%    STRESS(1)  x                STRAIN(1)  x
%    STRESS(2)  y                STRAIN(2)  y
%    STRESS(3)  xy (SHEAR)       STRAIN(3)  xy (SHEAR STRAIN)
  TYPE SJOINT E(STRESS(3)) I(STRAIN(3))
```

illustrated in Figure 3.3, which represents the same storm sewer model as Figure 2.1. The terminals are all of type FLUID, which is defined by the statement

```
TYPE FLUID E(B) I(FLOW)
```

FLOW represents the fluid flow in volume/unit time. B represents the total energy of the fluid at a given point (terminal) and is defined as the sum of the kinetic, potential, and intrinsic (pressure) energy of the fluid.

The local terminals of element SEWER (which are used to interconnect the external terminals of its subelements) are numbered 1 through 9. The remainder of Figure 3.3 depicts the subelements of SEWER. Instances of a new element, called CAP, are used to close unused outlets by setting the flows through them to zero in an equation of element CAP. The

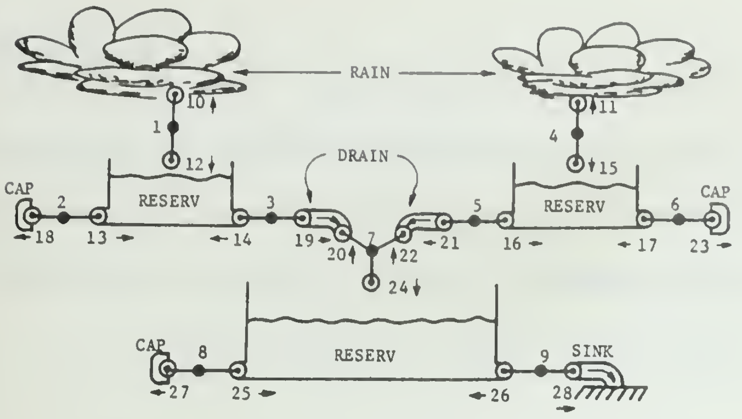


Figure 3.3. Storm Sewer Network

external terminals of the subelements of SEWER are numbered 10 through 28 for the purpose of illustration. The positive direction of FLOW at each of these external terminals, under the sign convention described earlier, is indicated by an arrow below the terminal.

The E-set node relationships, which are applied automatically, can be summarized by the equalities below:

```
B(1) = B(10) = B(12)
B(2) = B(13) = B(18)
B(3) = B(14) = B(19)
B(4) = B(11) = B(15)
B(5) = B(16) = B(21)
B(6) = B(17) = B(23)
B(7) = B(20) = B(22) = B(24)
B(8) = B(25) = B(27)
B(9) = B(26) = B(28)
```

The I-set node relationships, which are applied automatically, can be summarized by the equations below:

```
FLOW(10) + FLOW(12) = 0
FLOW(11) + FLOW(15) = 0
FLOW(13) + FLOW(18) = 0
FLOW(14) + FLOW(19) = 0
FLOW(16) + FLOW(21) = 0
FLOW(17) + FLOW(23) = 0
FLOW(20) + FLOW(22) + FLOW(24) = 0
FLOW(25) + FLOW(27) = 0
FLOW(26) + FLOW(28) = 0
```

Once again it should be noted that local terminals do not enter into the

I-set node relationship. In fact, local terminals do not even have associated I-sets because no meaningful sign convention exists for I-set members at local terminals which are connected to two or more external terminals (as is typically the case). Thus the equations of an element may not reference I-set members at local terminals.

3.5. Subelement Invocation and Connections

Subelements are invoked by the SUB-ELEMENT statement, which has the general form

SUB-ELEMENT _name_ [_parm_=_exp_;]*

name represents the name of the element to be invoked as a subelement. _parm_=_exp_ represents a non-default assignment for a parameter, _parm_, of the subelement. _exp_ can be any well defined MODEL expression whose dimensions are the same as _parm_'s and whose variables are known within the invoking element. Non-default parameter assignments may be specified in any order and they may be omitted when default parameter assignments are defined for the subelement.

A single SUB-ELEMENT statement is used for each subelement invocation. When many non-default parameter assignments are needed, they can be continued on the following line(s) or the NON-DEFAULTS statement can be used. This is the complement of the DEFAULTS statement and has the general form

NON-DEFAULTS [_parm_=_exp_;]*

where _parm_=_exp_ is as described above. Any number of NON-DEFAULTS statements can be used. Any parameters which are not assigned values in SUB-ELEMENT or NON-DEFAULTS statements will assume their default values if those are defined.

In order to accommodate multiple invocations of the same element, each NON-DEFAULTS statement used must follow the SUB-ELEMENT statement to which it refers and no other SUB-ELEMENT statements may intercede.

The use of the ELEMENT, DEFAULTS, SUB-ELEMENT and NON-DEFAULTS statements to assign parameter values for subelement instances is illustrated by the following example. The element EX has parameters which are defined by the statements

```
ELEMENT  EX  P1 ; P2 = 5 ; P3(4,4)
DEFAULTS  P4 = 4 ; P5
```

It is invoked by an element which includes the statements

```
LOCALS  X ; Y ; A(4,4)
SUB-ELEMENT  EX  P1 = X + Y ; P3 = A
NON-DEFAULTS  P5 = 21 ; P2 = 7
```

For this instance of EX, P1 will assume the value of X + Y, the array P3 will be equivalent to the array A, P5 and P2 will assume the values 21 and 7, respectively, and P4 will assume its default value, 4.

Subelements are connected into a network by the SUB-CONNECT statement, which has the general form

SUB-CONNECT [<u>id</u>]*

[<u>id</u>]* represents a list of the ID numbers of the terminals in the invoking element to which the external terminals of the subelement are to be connected. This list must include one terminal ID number for each external terminal of the subelement and the order of this list must correspond to the order in which the external terminals of the subelement are defined in its TERMINALS statement(s).

For example, the statements

```
ELEMENT  SEWER
     .
TERMINALS  1-9 FLUID
     .
SUB-ELEMENT  RESERV
SUB-CONNECT  4 5 6
     .
ELEMENT  RESERV
     .
TERMINALS  0-2 FLUID
EXTERNALS  0-2
```

cause terminals 4, 5 and 6 of element SEWER to be connected, respectively, to external terminals 0, 1 and 2 of subelement RESERV. If the TERMINALS statement in element RESERV is changed to

```
TERMINALS  2 1 0
```

terminals 4, 5 and 6 of element SEWER are connected, respectively, to external terminals 2, 1 and 0 of subelement RESERV.

The external terminals of a subelement may be connected to any terminals, local or external, of the invoking element.

Like the NON-DEFAULTS statement, the SUB-CONNECT statement must follow its corresponding SUB-ELEMENT statement with no other SUB-ELEMENT statements interceding. Multiple SUB-CONNECT statements may be used in conjuction with the invocation of a subelement which requires many connections.

The SUB-CONNECT statement can also be used to establish a _pseudo-ID number_ for an external terminal of a subelement. This would be done to allow the equations of the invoking element to reference an I-set terminal variable at that external terminal of the subelement. (References to E-set terminal variables do not present the same problem because these variables have the same value at the terminal of the invoking element to which they are connected.) A pseudo-ID number to be used for such a reference is established by attaching it with a minus sign to the ID number of the connected terminal in the SUB-CONNECT statement. For example, the statements

```
SUB-ELEMENT  Y
SUB-CONNECT  5 1-2 3
```

define the connections for subelement Y and establish -2 as the pseudo-ID number of the second external terminal of Y. The equations of the invoking element may then reference a member of the I-set, say FLOW, at this external terminal of the subelement instance in the form FLOW(-2).

When a pseudo-ID number is used to reference an I-set variable at an external terminal of a subelement, the sign convention discussed in section 3.1 above applies. The I-set variable represents a quantity directed into the subelement. Thus FLOW(-2) is positive if the flow is directed into the subelement and is negative if the flow is directed out the subelement.

In most networks the necessary terminal connections are specified by SUB-CONNECT statements. However, the local and external terminals of an element may be freely interconnected with each other when desired. This is done by the CONNECT statement, which has the general form

CONNECT [<u>id1</u>-<u>id2</u>]*

where <u>id1</u> and <u>id2</u> represent ID numbers of terminals to be connected.

3.6. <u>Strip Terminals</u>

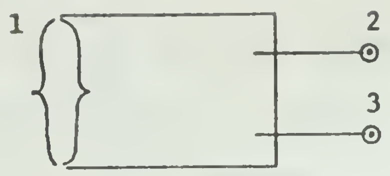
In some networks it may be useful to have different numbers of external connections for different instances of the same element. For example, an element which computes the sum and the maximum of a variable number of input lines remains the same functionally regardless of the number of inputs which are connected to it. If it is modeled using ordinary external terminals, a new element definition must be specified for each different number of external connections to be used or the different input lines must be shorted together at one external terminal. The first approach is very inconvenient and the second approach does not allow the voltage summer to operate properly. However, a special kind of external terminal, called a _strip terminal_, allows a single element definition to be used with a variable number of external connections without shorting any connections together. Any number of terminals may be directly connected to an external strip terminal of a subelement. Each such connection causes MODEL to generate a unique terminal within the subelement to receive the connection. This allows a device such as the voltage summer to be modeled by a single element, regardless of the number of external connections used for each invocation. In Figure 3.4, element SUMMER is defined using a strip terminal.

Strip terminals are defined in the TERMINALS statement by prefixing their ID numbers with the letter S. Thus the statement

```
TERMINALS  S1 2-3
```

defines terminal 1 to be a strip terminal. Strip terminals must be defined singly, rather than in consecutively numbered groups. Strip terminals always have external scope and need not be defined as external in EXTERNALS statements.

Element SUMMER has three external terminals, defined by the statements



{} represents strip terminal

```
ELEMENT SUMMER
TERMINALS S1 CONTROL 2-3 CONTROL
EXTERNALS 2-3
EQUATIONS LEVEL(2)=MAX(LEVEL(1))
          LEVEL(3)=SUM(NSTRIP(1),LEVEL(1))
```

Figure 3.4. Voltage Summer Element.

```
TERMINALS  S1 CONTROL 2-3 CONTROL
EXTERNALS  2-3
```

The terminal type CONTROL is defined by the statement

```
TYPE  CONTROL E(LEVEL)
```

For strip terminals to be useful there must be some special operators available for referencing the generated terminals in the equations of the subelement. The possibilities which exist are illustrated by the equations of element SUMMER. The input lines to the voltage summer are connected to the strip terminal. External terminals 2 and 3 transmit the maximum and the sum, respectively, of the input levels. This is expressed in the equations of SUMMER in the form

```
LEVEL(2) = MAX(LEVEL(1))
LEVEL(3) = SUM(NSTRIP(1),LEVEL(1))
```

In the first equation above MAX refers to a built-in function which computes the maximum of a variable number (from 1 to 10) of arguments.

In the second equation, SUM refers to a user-written FORTRAN function subprogram (see section 2.15), which computes the sum of a variable number of arguments.

References to strip terminal variables such as LEVEL(1) are treated in a special way when they appear as function arguments as above. LEVEL(1) will be replaced in the argument lists above by the sequence of arguments

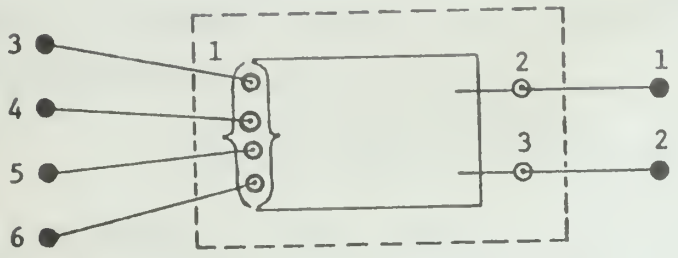
LEVEL(3),LEVEL(4),LEVEL(5),LEVEL(6)

where 3, 4, 5 and 6 are the ID numbers of the terminals generated by the connection of terminals 3, 4, 5 and 6 of the invoking element to the strip terminal of SUMMER.

In the second equation above, NSTRIP is a built-in function whose value at a strip terminal is the number of terminals directly connected to that strip terminal. It is used to indicate the number of actual arguments present for each reference to the user-written function SUM. (This is not necessary for the built-in function MAX.) In this case, strip terminal 1 has four direct connections so NSTRIP(1) has the value four.

The use of these special operators for strip terminal variable references allows the equations above to model the behavior of the voltage SUMMER as desired. The use of NSTRIP in the second equation also illustrates that other arguments may precede (or follow) a set of generated terminal variables in a function reference.

Strip terminals of subelements are connected into networks by CONNECT statements in the invoking elements, as shown in Figure 3.5.



```
SUB-ELEMENT SUMMER
SUB-CONNECT S7 1 2
CONNECT 3-7 4-7 5-7 6-7
```

Figure 3.5. Voltage Summer Connected as a Sub-element.

To permit these terminals to be identified in the CONNECT statements of an invoking element it is necessary to establish a unique pseudo-ID number for each strip terminal of each subelement invoked by an element. This is done in SUB-CONNECT statements by entering S<u>n</u>, where <u>n</u> is a unique ID number, in place of a connection to the invoking element. For example, the statements

```
SUB-ELEMENT  SUMMER
SUB-CONNECT  S7 1 2
```

specify that the first external terminal of subelement SUMMER is a strip terminal which will be referenced in the CONNECT statements of the invoking element as terminal 7.

The SUB-CONNECT statement shown above does not define a connection for the external strip terminal as it does for the other external terminals of subelement SUMMER. Connections to the strip terminal are specified in CONNECT statements of the invoking element. For example, the statement

```
CONNECT 3-7 4-7 5-7 6-7
```

causes the generation of four external terminals in subelement SUMMER and connects these terminals to terminals 3, 4, 5 and 6, respectively, of the invoking element.

A strip terminal of a subelement may be connected to any number of local and external terminals in the invoking element. However, connections between two strip terminals or between a strip terminal and another terminal of the same element are not allowed.

3.7. Equation-Variable Relationships in Network Models

In any well defined model there will be one equation for each variable present. The variables of a model consist of the parameters, global and local variables, and terminal variables of each element instance used. The equations which define the behavior of these variables are the equations and parameter assignments of each instance and the E-set and I-set terminal variable relationships at each node.

The behavior of parameters is usually defined by the parameter assignments used in invoking subelements (e.g., P=5, P=X+Y*Z). (Parameters of the highest level element of a model are assigned in the run control program, which is discussed in section 4.3.) Sometimes parameters of an element represent state or auxiliary variables whose behavior is defined by equations of the element. In these cases parameters will simply be replaced by variables of the invoking element (e.g., PR = R) and the invoked element will include additional equations which define these parameters. For example, instances of a subelement called POLAR might be used to define the polar coordinates (R, THETA) of points whose cartesian coordinates (X,Y) are defined elsewhere. Element POLAR would have four parameters; PX, PY, PR and PTHETA, and would consist of the two equations

```
PR = SQRT(PY**2 + PX**2)
PTHETA = ATAN(PY/PX)
```

which define the variables which replace PR and PTHETA for each invocation.

In general, each element should include one equation for each global and local variable defined in the element. However, a global or local variable can be defined in one element while its behavior is defined by an equation of a subelement which references the global variable or is invoked with the local variable as a parameter value as described above.

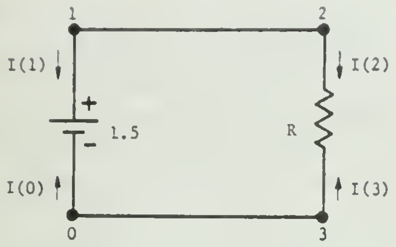
As concerns terminal variables, each distinct E-type and I-type terminal variable of the network must be defined by an equation in some element. The manner in which equations defining terminal variable behavior are associated with elements depends on the meaning of the elements--there is no rigid general rule. The two most common fashions of defining terminal variable behavior are described in the following two paragraphs.

Functional elements which have distinct input and output connections typically include one equation for each terminal variable at each external 'output' terminal. In these cases one and only one 'output' terminal will be connected to each node. The voltage summer described in the previous section is an example of such an element.

Elements which make no distinction between input and output terminals (e.g., the resistor element described in the previous section and the reservoir element of the storm sewer model) typically include one equation for each E/I-set pair at each external terminal. (There is only one equation for each pair of terminal variables because the node relationships for connected terminals provide additional equations defining terminal variable behavior.) In these cases an element with M external terminals, each having P pairs of E/I-set variables, will typically include MP equations defining terminal variable behavior. In elements with multiple external terminals (M>1) there is typically one equation for each member of the I-set. This equation relates the values of that I-set variable at all of the external terminals--often by summing these values to zero. For each E/I-set pair the remaining M-1 equations relate the values of the E-set variable at the M external terminals.

The analysis of electrical and structural networks frequently leads to one or more redundant node equations and one or more 'floating' variables. In this case the number equations will not equal the number of variables. For example, the simple electrical circuit in Figure 3.6 involves four voltages and four currents--one voltage/current pair at each of the terminals 0-3.

There are eight equations present of which the first four below come from the definitions of the battery and resistor elements and the remaining four come from the two terminal connections. However, this system of eight equations in eight variables can be simplified by eliminating the voltages V(2) and V(3) with equations 5 and 7 and eliminating the currents I(1), I(2), and I(3) with equations 2, 6 and 4.



1. V(1)=V(0)+1.5
2. I(0)+I(1)=0
3. V(3)=V(2)-I(2)*R
4. I(2)+I(3)=0
5. V(1)=V(2)
6. I(1)+I(2)=0
7. V(0)=V(3)
8. I(0)+I(3)=0

Figure 3.6. Example of Redundant Node Equations.

This simplification yields the system

9. V(1) = V(0) + 1.5
10. V(0) = V(1) - I(0)*R

in which one of the voltages is free to 'float', and the redundant node equation

8. I(0) + I(3) = 0

which is implied by equations 2, 6 and 4. MAP will automatically perform simplifications like those above and will also recognize and remove redundant equations such as equation 8. Thus MAP would reduce this trivial network to a system of two equations (9 and 10) in three variables (V(1), V(0) and I(0)). However, this does not mean that the network model is in error. There is an extra degree of freedom because no voltage reference level (ground) was specified for the model. The numerical software will fix one of the voltages at zero, effectively reducing the network to a system of two equations in two variables.

It is worth noting that if V(0) is fixed in the model (say at zero), MAP will reduce this network to two assignment statements to the effect

V(1) = 1.5
I(0) = 1.5/R

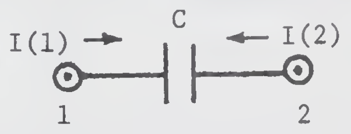
which need only be executed once.

3.8. Examples of Network Elements

The remainder of this section presents four examples of element definitions which illustrate the use of equations to define terminal variable behavior in the fashion described in the preceding sections. These examples come from the areas of electrical, mechanical and fluid networks. The elements to be modeled are a capacitor, a spring, a reservoir and a drain.

Capacitor

A graphical representation of a capacitor and the statements which define this element are shown in Figure 3.7.



```
TYPE  ELEMENT E(V) I(I)
```

```
TYPE  ELECTRIC E(V) I(I)
ELEMENT  CAPACITR  C
TERMINALS  1-2 ELECTRIC
EXTERNALS  1-2
EQUATIONS  I(1) + I(2) = 0
           C*(V(1)' - V(2)') = I(1)
```

Figure 3.7. Capacitor Element.

The capacitor has one parameter, C, which is its capacitance. Terminals 1 and 2 are external terminals of type ELECTRIC with E-set consisting of V, the voltage, and I-set consisting of I, the current. The first equation states that any applied current is transmitted through the capacitor. The second equation relates the voltage across the capacitor to the current through it according to the capacitor equation

C*V' = I.

The equation is expressed as if positive current enters at terminal 1--the voltage across the capacitor is expressed as V(1) - (V2) and the current through the capacitor is expressed as I(1). This yields the equation

C*(V(1)' - V(2)') = I(1).

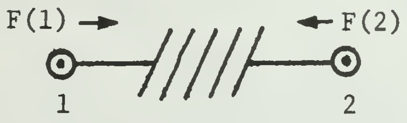
However, since I(1) = -I(2), this is equivalent to the equation

C*(V(2)' - V(1)') = I(2).

which is the natural form when positive current enters at terminal 2. This element will model capacitor behavior properly regardless of the direction of current flow. The capacitor element could be connected to any similarly modeled electrical elements such as resistors, batteries, etc.

Spring

A graphical representation of a spring and the statements which define this element are shown in Figure 3.8.



```
TYPE  MECH  E(X) I(F)
ELEMENT  SPRING  K ; L
TERMINALS  1-2 MECH
EXTERNALS  1-2
EQUATIONS  F(1) + F(2) = 0
           K*(X(2)-X(1)-L) = F(2)
```

Figure 3.8. Spring Element.

The spring has two parameters, K and L, which are the spring constant and the undeformed length, respectively. Terminals 1 and 2 are external terminals of type MECH with E-set consisting of X, the displacement, and I-set consisting of F, the applied force. The first equation states that applied forces are transmitted through the spring. The second equation relates the deformation of the spring to the force exerted by it according to Hooke's law

$$K*X = F.$$

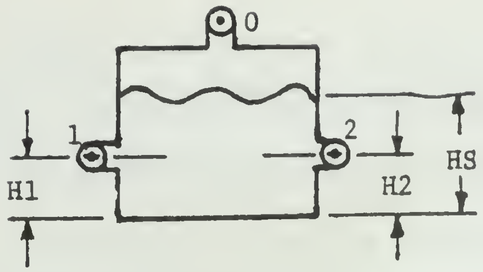
The positive x axis is assumed to be in the direction from terminal 1 toward terminal 2. Thus the deformation of the spring is expressed as X(2) - X(1) - L. The associated force exerted by the spring is F(2) whose positive direction is from terminal 2 toward terminal 1. Hence the equation

$$K*(X(2)-X(1)-L) = F(2)$$

This spring element could be connected to any similarly modeled mechanical elements such as other springs, fixed and moveable masses, etc.

Reservoir

A graphical representation of the reservoir used in the storm sewer model of Figure 2.1 and appendix B and the statements which define this element are shown in Figure 3.9. Terminals 0, 1 and 2 are external terminals of type FLUID with E-set consisting of B, the total energy of the fluid, and I-set consisting of FLOW, the fluid flow rate (in volume/unit time). Fluid enters the reservoir at terminal 0 (e.g., in the form of rainfall) and flows out of the reservoir through the outlets at terminals 1 and 2. The reservoir has five parameters, AS, A1, H1, A2 and H2, which specify the area of its surface and the cross-sectional areas and heights of the outlets at terminals 1 and 2. Local variables HS, V1, P1, V2, and



```
TYPE  FLUID  E(B) I(FLOW)
ELEMENT  RESERV  AS ; A1 ; H1 ; A2 ; H2
TERMINALS  0-2 FLUID
EXTERNALS  0-2
LOCALS  HS ; V1 ; P1 ; V2 ; P2
EQUATIONS  HS'*AS = FLOW(0) + FLOW(1) + FLOW(2)
           V1 = FLOW(1)/A1
           V2 = FLOW(2)/A2
           B(1) = V1*V1/(2*G) + P1/GAMMA + H1
           B(2) = V2*V2/(2*G) + P2/GAMMA + H2
           B(1) = B(0)
           B(2) = B(0)
           B(0) = HS
```

Figure 3.9. Reservoir Element.

P2 are used to express the height of the reservoir's surface and the fluid velocities and pressures at the two outlets. The reservoir element also utilizes global variables G and GAMMA which represent the acceleration due to gravity and the density of the fluid.

The first equation defines the rate of change in the height of the reservoir's surface, HS', in terms of the rates of flow into and out of the reservoir. By convention the rate of flow into the reservoir is FLOW(0) and the rate of flow out of the reservoir is -(FLOW(1) + FLOW(2)). The second and third equations define the velocities, V1 and V2, of the fluid flowing through the two outlets (in distance/unit time) in terms of the rates of flow.

The remaining equations define the fluid pressure at terminals 1 and 2 (P1 and P2), and the total energy of the fluid at terminals 1, 2 and 0 (B(1), B(2) and B(0)). They make use of the Bernoulli equations of fluid mechanics applied to a steady flow situation. These equations state that the quantity

$$B = V^2/2G + P/GAMMA + H$$

has the same value at any point along a flow line. The variables in this quantity are

B	the total energy of the fluid
G	the acceleration due to gravity
GAMMA	the density of the fluid
V	the velocity of the fluid
P	the pressure of the fluid (above one atmosphere)
H	the height of the point in question

The equations

```
B(1) = V1*V1/(2*G) + P1/GAMMA + H1
B(2) = V2*V2/(2*G) + P2/GAMMA + H2
```

express the total energy of the fluid at terminals 1 and 2 (B(1) and B(2)) in terms of the pressures, velocities, and heights of the fluid at those terminals. These equations serve to define P1 and P2 since all other quantities in these equations are defined elsewhere. The equations

```
B(1) = B(0)
B(2) = B(0)
```

equivalence the values of the Bernoulli quantity at terminals 1 and 2 to its value at the reservoir's surface.

The equation

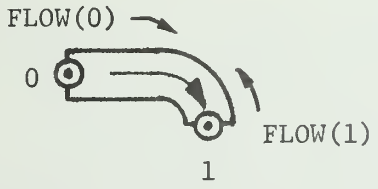
```
B(0) = HS
```

defines the value of the Bernoulli quantity at the reservoir's surface to be equal to its height. The velocity and pressure terms are ignored here because the velocity, HS', is assumed to be too small to affect the Bernoulli quantity appreciably (due to the reservoir's very large surface area) and the pressure is assumed to be one atmosphere.

This reservoir element could be connected to any similarly modeled fluid elements such as other reservoirs and pipes or the drain element which is described next.

Drain

A graphical representation of the drain used in the storm sewer model of Figure 3.3 and appendix B and the statements which define this element are shown in Figure 3.10.



```
TYPE  FLUID  E(B) I(FLOW)
ELEMENT  DRAIN  A ; H
TERMINALS  0-1 FLUID
EXTERNALS  0-1
EQUATIONS  FLOW(0) + FLOW(1) = 0
           FLOW(0) = ABS(SQRT(2*G*A*A*(B(0) - H)))
```

Figure 3.10. Drain Element.

The drain has two parameters, A and H, which specify its cross-sectional area and height. Terminals 0 and 1 are external terminals of type FLUID with E-set consisting of B, the total energy of the fluid, and I-set consisting of FLOW, the flow rate (in volume/unit time). Fluid flows into the drain through a connection at terminal 0 (e.g., from a reservoir) and flows out of the drain through a connection at terminal 1 (e.g., to another reservoir at atmospheric pressure.)

The first equation states that all fluid flowing into the drain flows out of the drain. The second equation defines the rate of flow through the drain in terms of the area and height of the drain and the total energy of the fluid. This equation was obtained by substituting FLOW(0)/A for V and 0 (i.e., one atmosphere) for P in the expression for the Bernoulli quantity, B(0), and solving for FLOW(0). The absolute value function is used to ensure that FLOW(0) is non-negative.

4. THE RUN CONTROL PROGRAM

The simulation model is controlled by a FORTRAN main program which calls NSP as a subroutine. This run control program (RCP) can be as simple or as sophisticated as the user's understanding of FORTRAN permits. The RCP sets the control variables (section 4.2) and the run parameters (section 4.3) and can reference variables of the highest level element of the model by their MODEL names. The RCP can also utilize any other user-supplied FORTRAN code, which can be passed through MAP or interfaced at the FORTRAN or load steps of the simulation (sections 4.4 through 4.6).

MAP produces the RCP from the run control file (section 4.1) following the model definition. If no run control file is provided, the resulting RCP will assign default values to the control variables and call NSP to solve for both the steady-state and transient solutions. Most simulations will require at least a few non-default values for the control variables. The run control file allows the user to tailor the RCP to his particular requirements.

4.1. Run Control File

The run control file contains the information that the MODEL compiler needs to construct the RCP. It consists of FORTRAN statements and MODEL run control keywords and is placed in a different part of the input deck than the model definition. No FORTRAN knowledge is required to create the RCP for single run simulation models. Multiple run simulations can be easily created by FORTRAN programmers and interfaced to other procedures using the run control keywords $NORCP and $DECLARE, as explained in section 4.4. FORTRAN functions and subroutines may also be included in the run control file for compilation as a part of the simulation model.

The RCP for a single run simulation assigns values to the control variables and run parameters and then calls NSP as a subroutine to perform the simulation. All control variables have preassigned default values, as shown in Tables 4.1 and 4.2. The default values may change according to the needs of the simulation as discussed in section 4.2. Run parameters do not have default values and must be assigned by the user.

4.2. Control Variables

The program control variables and the simulation control variables provide communication between the RCP and NSP. These control variables reside in common storage so they may be accessed by any subprogram which contains the appropriate COMMON statements, as described below. All control variables are initialized to default values by RCP. Thus, only variables which require non-default values need to be included in the run control file. To change the default value of a control variable, the statement

<u>var</u> = <u>value</u>

must be included in the run control file, where <u>var</u> is a control variable name and <u>value</u> is a decimal or integer constant. For example, the statement

```
TSTOP = 25.0
```

changes the final value of the independent variable from the default of 1.0 to a value of 25.0.

Certain restrictions have been imposed on the format of control variable and run parameter assignment statements for FORTRAN compatibility. The control variable or run parameter name must always be to the left of the equal sign because the statement is an assignment instruction and not a mathematical equality. Each assignment statement must be put on a separate line and indented at least six spaces. The values are assigned in the order in which they appear in the run control file and do not change during the simulation, except for ISTOP, which returns the termination code for the run.

Table 4.1. Program Control Variables.

Name	Description	Default
ISNAP	Diagnostic print control 0 - no diagnostic print 1 - limited trace and debug information 5 - prints everything (use with care)	ISNAP = 0
MODE	Solution mode 0 - steady-state and transient solution 1 - steady-state solution only 2 - transient solution only	MODE = 0
IREAD	Logical unit number for input	IREAD = 5
IWRITE	Logical unit number for printed output	IWRITE = 6
ISAVE	Logical unit number for stored output	ISAVE = 1

The program control variables and default values are listed in Table 4.1. The simulation control variables and defaults are listed in Table 4.2. The control variables are stored in labelled COMMON and may be accessed in FORTRAN subprograms by including the declarations below.

```
      COMMON/SWITCH/ISNAP,MODE,IREAD,IWRITE,ISAVE
      COMMON/SYSPRM/HMIN,HMAX,EPS,DEL,TSTART,TSTOP,COMINT,ISTOP
```

Common block SWITCH contains the program control variables and the simulation control variables are in common block SYSPRM. Detailed explanations of the control variables are given below.

Table 4.2. Simulation Control Variables.

Name	Description	Default
HMIN	Minimum integration stepsize in transient solution.	HMIN = .000001
HMAX	Maximum integration stepsize in transient solution	HMAX = 1.0
EPS	Relative error in transient solution	EPS = .00001
DEL	Absolute error in steady-state solution	DEL = .00001
TSTART	Initial value of independent variable in transient solution	TSTART = 0.
TSTOP	Final value of independent variable in transient solution	TSTOP = 1.0
COMINT	Interval at which transient solution is reported	COMINT = .01
ISTOP	Termination code -n - NSP error n 0 - Normal termination +n - STOP condition n	ISTOP = 0

ISNAP--Controls the diagnostic printout from NSP. When ISNAP = 1, the values of all variables are printed after each iteration of the steady state analysis and after each integration step of the transient analysis. The higher diagnostic levels shown in Table 4.1 are useful mainly for isolating system errors. ISNAP can also be useful for debugging user-supplied routines.

MODE--Selects the type of analysis to be performed. MODE = 0 specifies both steady-state and transient analysis, as described in sections 1.2 and 1.3. The independent variable assumes its initial value (TSTART) and the initial values specified in the model are assigned. The uninitialized variables and derivatives are adjusted to obtain the steady-state solution and the transient analysis then begins, proceeding until the terminal value of the independent variable (TSTOP) is reached or a STOP condition specified in the model is encountered.

MODE = 1 specifies steady-state analysis only. A steady-state solution of the model is calculated as described above, but no transient analysis is performed. In this case, TSTART and DEL are the only control variables whose values need to be specified.

MODE = 2 specifies transient analysis only. No initial state is calculated. The simulation proceeds from a state established by user-supplied code which initializes the model variables ZT, ZG, ZY, and ZD. The MODEL names for these variables may be obtained from the model dictionary as explained in section 5.2. This mode may also be used to restart the transient analysis after a termination condition is encountered. In this case, the terminal state of the model variables becomes the new initial state. However, before the transient analysis is restarted, the condition which caused termination must be changed. Otherwise, the transient analysis will terminate immediately.

IREAD, IWRITE, ISAVE--Logical unit numbers for input, printed output and stored output, respectively. Non-default values for unit numbers require changes to the RCP name card and the job control statements.

HMIN--The minimum stepsize in the transient solution. HMIN determines the smallest error that can be attained in the transient solution (EPS) and may be estimated from the inequality

$$\mathrm{HMIN} \le \mathrm{EPS}/|(Y''/Y)|,$$

where Y is the solution vector. HMIN is also the error allowed in the location of state transitions in conditional equations. It expresses the minimum interval of interest in the independent variable.

The default initial stepsize is set to

```
H = 10**((log(HMIN) + log(HMAX))/2)
```

Problems with decaying periodic components often require the minimum stepsize at the start of the simulation. If HMIN < 0, the initial stepsize is set to |HMIN| and the minimum stepsize is |HMIN|.

HMAX--The maximum stepsize in the transient solution. HMAX affects the initial stepsize (see HMIN) and may be used to restrict the stepsize computed by the error control mechanism. Problems which initially have a slowly changing solution may be started more efficiently if the initial stepsize is set to HMAX. When HMAX < 0, the initial stepsize is set to |HMAX| and the maximum stepsize is |HMAX|.

EPS--The maximum relative error introduced at any step in the transient solution. For smooth problems with continuous high order derivatives, the maximum relative error in the computed solution will be roughly proportional to EPS. The constant of proportionality depends on the magnitude of the derivatives in the solution and the length of the interval on which the solution is computed. The error constant may be estimated experimentally by comparing the computed solutions at various values of EPS.

DEL--The maximum absolute error in the steady state solution. DEL must be small enough to produce an initial state with relative error less than EPS. If this condition is not met, the error in the transient solution will always exceed EPS on the first step.

TSTART--The initial value of the independent variable in the transient solution.

TSTOP--The final value of the independent variable in the transient solution.

COMINT--The interval in the independent variable at which the transient solution is reported. Looking ahead one step, the stepsize is adjusted to coincide with the reporting points. The value of COMINT is determined by the intended use of the output. The total number of output reports is (TSTART-TSTOP)/COMINT+1. If COMINT equals zero, every step of the transient solution is reported.

ISTOP--The termination code returned from each simulation run. If the simulation terminates normally, ISTOP is set to zero. If an error occurs, ISTOP is set to -n, where n is an NSP error code. If the transient solution is terminated by a conditional stop, the STOP code is returned in ISTOP. ISTOP must be zero at the start of each simulation run. For multiple run simulations ISTOP is especially useful for determining the control sequence between runs.

4.3. Run Parameters

Run parameters represent physical constants in the simulation model whose values may be varied for each simulation run. The run parameters are the element parameters of the highest level element in the model (section 2.4). The RCP does not assign default values to run parameters. Therefore, each run parameter must be given a value by including a statement of the form

parm = _value_

in the run control file. The MODEL names of the run parameters are used for _parm_ and _value_ is a decimal constant. Parameter assignment statements must conform to the same FORTRAN restrictions described for control variable assignment statements in section 4.2.

For example, if TOP is the highest level element in a model, the MODEL statement below defines ALPHA and BETA as run parameters.

```
ELEMENT TOP ALPHA;BETA
```

To assign the value 32.5 to ALPHA and the value .015 to BETA, the statements below should be included in the run control file.

```
ALPHA = 32.5
BETA = 0.015
```

The run parameters are stored in common block VARS, which is automatically supplied by the MODEL compiler, as discussed in section 4.4. They may also be accessed by their FORTRAN names in any subprogram which includes the appropriate common block declaration. The FORTRAN names can be found in the model dictionary, as explained in section 5.3, or by inspecting the equivalence statements produced by MAP in the RCP.

4.4. <u>FORTRAN</u> <u>Run</u> <u>Control</u>

Multiple run simulations and interfaces to other FORTRAN procedures can easily be created by passing user-written control routines through MAP, which automtically supplies the FORTRAN declarations for the FORTRAN variables and arrays required by the simulation model. In addition, equivalence statements are produced that allow the variables and parameters of the highest level element to be referenced by their MODEL names. MAP also produces default value assignments for the control variables.

The run control keywords $NORCP and $DECLARE control the generation of the RCP and the location of the model declarations in the RCP. If neither of these keywords is used, the default RCP shown in Figure 4.1 is produced by the MODEL compiler. Note that the user's run control file appears immediately after the control variable defaults and just before the call to invoke NSP. This allows the statements in the run control file to redefine control variables and make parameter assignments before NSP is called. In this case, the run control file can contain only executable FORTRAN statements and FORMAT statements. The reason for this is that most FORTRAN compilers require declaration statements to be placed before the first executable statement. The control variable default value assignments are the first executable statements and therefore all subsequent statements must be executable, with the exception of FORMAT statements, which can be placed anywhere in a program unit.

```
<Program name statement (CDC version only)>
<Declarations for model variables>
<Declarations for variables and parameters
 of the highest level element in the model>
<Equivalences between MODEL and FORTRAN names>
<Control variable default value assignments>
<User's run control file>
10  CALL NSP
    END
```

Figure 4.1. Default Run Control Program.

The MODEL names of the variables and parameters of the highest level element in the model are all declared to be double precision variables. Other FORTRAN variables are assigned types according to the first letter of each variable name. Variables beginning with the letters A-H and Q-Z are double precision. Variables beginning with the letters O and P are single precision and variables beginning with the letters I-N are integers. To define variables with non-default types the user must use the $NORCP option described below.

A statement number is provided on the default RCP statement which calls NSP for the purpose of creating simple multiple run simulations. For example, the run control file in Figure 4.2 assigns non-default values to TSTOP and COMINT, then defines a DO loop which repeatedly reads input data and calls the simulation model. The data contains a run identification number and parameter values for ALPHA and BETA, which are printed for each run. The simulation loop stops if the loop limit of 25 is reached, or if the run number is less than or equal to zero.

The default RCP offers considerable flexibility and should be adequate for most simulation models which do not need to be interfaced to other

```
      TSTOP = 100.0
      COMINT = 10.0
      DO 10 I = 1,25
         READ(5,1) ID,ALPHA,BETA
1        FORMAT(I2,2F10.2)
         IF(ID.LE.0) STOP
         PRINT(6,2) ID,ALPHA,BETA
2        FORMAT(9H RUN ID =,I3,
     +          5X,7HALPHA =, F10.2
     +          5X,6HBETA =, F10.2)
```

Figure 4.2. Run Control File for Multiple Run Simulation.

FORTRAN procedures. When the simulation model is to be called by other procedures, as in the case of optimization or parameter estimation studies, the $NORCP option allows the user to define a non-default RCP and also to pass other FORTRAN subrograms through MAP for compilation with the simulation model.

When $NORCP is used, it must be the first line of the run control file. $NORCP informs MAP that the user has supplied a run control program and the default RCP is suppressed. The line following $NORCP in the run control file must be the first line of a main program unit or a subprogram name statement. Thereafter, FORTRAN statements can be used without restriction to define the RCP and any additional subprograms that are required.

When the $DECLARE keyword is present in the run control file, it instructs the MODEL compiler to replace the keyword by the model declarations, equivalences, and default assignments in the resulting RCP. As before, only executable statements may follow the $DECLARE in the subprogram where it appears because the default assignments are executable statements. But now, user declarations which precede a

$DECLARE can be merged with the model declarations to produce a legal sequence of statements.

$NORCP and $DECLARE should always be used as a pair. If $NORCP is used alone, the user has no a priori way of creating the necessary model declarations. If $DECLARE is used without $NORCP, it is ignored and a warning message is printed.

The use of $NORCP and $DECLARE to produce a simulation model that can be called as a subroutine is shown in Figure 4.3. The calling program supplies the parameters A and B to the subroutine, which returns the computed value of X. The MODEL names of these variables are ALPHA, BETA, and XI, respectively. MODEL names cannot be used as subroutine parameters in the subroutine where they are DECLAREd because they are located in COMMON storage. (Subroutine parameters are dummy variables which reference storage locations in the calling program. A conflict over the location of the parameter arises if it is also declared to be located in COMMON.) Consequently, the variables A, B, and X are defined to be the subroutine parameters. The model parameters, ALPHA and BETA, are assigned the values of A and B each time the subroutine is called. The result of the simulation, XI, is returned to the calling program by means of the variable X.

The variables A, B, and X are declared to be double precision for agreement with the model variables. This is not essential and would be undesirable if the calling procedure expects a single precision result.

In Figure 4.3, the model declarations and default assignments are substituted for the $DECLARE. Then the control variables TSTOP, COMINT,

```
$NORCP
      SUBROUTINE MODEL(A,B,X)
      DOUBLE PRECISION A,B,X
$DECLARE
      TSTOP = 100.0
      COMINT = 100.0
      IWRITE = 1
      ALPHA = A
      BETA = B
      CALL NSP
      X = XI
      RETURN
      END
```

Run control file to formulate the simulation model as a SUBROUTINE.

Figure 4.3.

and IWRITE are redefined. Since the simulation is to be used as a subroutine for another procedure, the printed output is suppressed by setting the print number to the save unit number. The quantity of saved output is reduced by setting COMINT equal to TSTOP.

4.5. Run-Control Programs Specified at the FORTRAN or Load Steps of NSP

When the run control program is not passed through MAP, it must be interfaced to NSP by including the FORTRAN or object code in the input file of the FORTRAN or load step, respectively, of NSP (section 4.6). In this case the FORTRAN declarations and common storage specifications mentioned in section 4.4 and the assignments of default or non-default values for all of the system control variables must be included by the user. However, MAP produces a default run control program when none is passed through MAP and the declarations, COMMON statements, and assignments of default control values can be copied exactly as they appear in the default run control. The default run control program can be listed for this purpose by including an SLIST (Short FORTRAN LIST) or an FLIST (Full FORTRAN LIST) statement before the first element of the input file.

If the user wishes to reference parameters and global and local variables of the highest level element by their MODEL names in a run control program which is not passed through MAP, he must include the EQUIVALENCE statements mentioned in section 4.4. These statements can also be copied exactly as they appear in the default run control program. Alternatively, the use of the model dictionary (section 5.2) allows the user to reference any MODEL variable or parameter by its FORTRAN name.

A default run control program looks very much like the initial portion of the run control programs for the examples in appendix B.

4.6. Modification of the FORTRAN Model at the Load Step of NSP

In addition to allowing FORTRAN code to be passed through MAP and automatically interfaced to NSP (see appendix A), MODEL also allows the user to interface FORTRAN code to NSP at its FORTRAN or load steps. This enables the user to interface new code (e.g., run control program, response testing procedures) without rerunning MAP or to replace any code produced by MAP (e.g., subroutine RECORD). This is done through the user-supplied job control which invokes the various steps of MODEL. The new FORTRAN code must be included in the input file of the FORTRAN step of NSP or it must be compiled and the resulting object code must be included in the input file of the load step of NSP. The specific job control which will accomplish this depends on the particular machine and installation to be used. Appendix C shows job control used to invoke MAP and NSP on an IBM 360 and CDC Cyber 175. User-supplied FORTRAN code can be included at the indicated places in the input deck. User-supplied binary files can be included by specifying those files in DD statements similar to the one used for &&LNKFILE in the IBM 360 NSP step and by adding additional load steps to the Cyber control statments before NSPBIN.

5. ERRORS AND DIAGNOSTICS

Errors may be detected in either the model analysis process or the numerical simulation process. When an error is detected, a message is printed and corrective action may be taken. This chapter explains the interpretation of the error messages and the use of diagnostic information produced by MAP to locate the cause of the error.

5.1. Model Analysis Diagnostics

The model analysis process provides extensive diagnostics concerning errors and possible errors in the model definition. MAP attempts to detect and report all possible errors in the model definition rather than stopping the model analysis after some fixed number of errors have been found. The model analysis is aborted only if its available storage becomes insufficient or if an element invokes itself as a subelement.

The diagnostics produced by MAP are classified as either warning messages or error messages. Warning messages describe trivial errors which MAP attempts to correct (e.g., names which are longer than eight characters are truncated by MAP) and situations which may or may not be actual errors (e.g., an unequal number of equations and variables). Error messages describe serious errors such as an equation with a missing operator or the connection of terminals of different types.

The simulation of a model will not be attempted if any error messages were produced during the model analysis. MAP does not attempt to correct nontrivial errors and allow NSP to simulate the corrected model because the execution of NSP, which is typically more expensive than the execution of MAP, would be useless if MAP made a wrong correcting assumption.

Since warning messages refer to situations which may have been corrected by MAP or which may not be actual errors, two options are available in response to these messages. By default the simulation of a model will be attempted if only warning messages were produced during model analysis. Alternatively, the user can cause the simulation to be aborted in response to warning messages by including the statement

ABORT

before the first element definition of the input file.

The model analysis is a four step process. Diagnostics may be produced during any of the first three steps, which are called the <u>input step</u>, the <u>expansion step</u>, and the <u>output step</u>. The fourth step organizes the FORTRAN output from the preceding steps into a single file suitable for compilation and execution with NSP. During the input step the model definition is read and the text statements are listed and translated into an internal representation. All errors and warnings which arise from statements which precede the first element definition are reported during this step. These messages are listed after the statement which is in error and the position within the statement at which the error was detected is indicated by a question mark below that position. In addition, a number of errors and warnings which arise from improper element definitions are reported during this step. Some of these messages are listed after the erroneous statement as described above. Others are listed after the last statement of the element definition.

In the latter case the erroneous portion of the statement in question is listed after the message. Each specific element definition error detected during the input step is reported only once, regardless of the number of instances of that element which are invoked.

During the expansion step the context of each instance is analyzed and each instance is expanded into a set of variables, equations, and initial values. All errors and warnings which arise from improper interconnection of instances or invalid syntax in equations, parameter assignments, or initial values are reported during this step. These messages are specific to particular instances so some of them may be repeated for each instance of a given element. The specific instance to which one of these messages refers is identified by its invocation list (see section 5.2), which is listed in the first line of the message. Each of these messages is followed by a line of text which clearly identifies the cause of the error. For example, a message concerning invalid equation syntax is followed by a listing of the equation in error and the position at which the error was detected is indicated by a question mark.

During the output step the expanded set of equations which represents the model is translated from an internal representation to a set of FORTRAN subroutines. Error and warning messages produced during this step concern the set of equations as a whole and are not specific to any particular instance. Each of these messages is followed by a line of text which clarifies the cause of the error or possible error. For example, the number of equations and the number of variables are listed following the warning message which states that these numbers are unequal.

The format of MAP error and warning messages is shown in Figure 5.1. Figure 5.1a shows a generalized error message. Warning messages differ from this general form only in that the word WARNING replaces the word ERROR on the first line. Besides indicating whether the message is a warning or an error message the first line identifies the instance involved (for expansion step messages) and gives the message number. Messages produced during the input and output steps list INPUT and OUTPUT, respectively, as the instance identification. Messages produced during the expansion step give the invocation list and instance number of the instance involved. This is represented in Figure 5.1a by _instance_. The message number is represented by _num_. It is included to aid the user in looking up the messages in appendix D of this manual. The second line of the message describes the particular error or possible error which was detected. This line is the same for every occurrence of a given message number. The third line of the message identifies the cause of the error or warning. This line will be different for each message. The third line is not included in input step messages which are listed immediately after the statement which caused the warning or error message.

Figures 5.1b through 5.1f are examples of diagnostic messages produced by MAP. Figure 5.1b shows the warning message caused by a terminal type name which is longer than eight characters. This message is produced during the input step and follows the TYPE statement which is in error. The position in this statement at which the error was detected is indicated by the question mark below the statement.

Figure 5.1. Format of Model Analysis Error and Warning Messages.

```
IN ELEMENT instance ERROR num
description of error
identification of cause
```

Figure 5.1a.

```
TYPE STRUCTURAL  E(DX;DY;DZ)  I(FX;FY;FZ)
                  ?
IN ELEMENT INPUT WARNING 13
TERMINAL TYPE NAME SHORTENED TO 8 CHARACTERS
```

Figure 5.1b.

```
ELEMENT
                ?
IN ELEMENT INPUT ERROR 26
ELEMENT NAME MISSING
```

Figure 5.1c.

```
IN ELEMENT INPUT ERROR 33
TERMINAL TYPE NOT ASSIGNED. TERMINAL TYPE NAME UNDEFINED
TERMINAL ID NUMBERS: 1-3, TERMINAL TYPE NAME: STRUCT
```

Figure 5.1d.

```
IN ELEMENT RADIO.AMP.RESISTOR(3) ERROR 43
EQUATION IGNORED.  OPERATOR MISSING BEFORE ?
V(1) - V(0) = I(1) R?#
```

Figure 5.1e.

```
IN ELEMENT OUTPUT WARNING 7
SYSTEM MAY BE SINGULAR. NUMBER OF EQUATIONS
NOT EQUAL TO NUMBER OF VARIABLES
NUMBER OF EQUATIONS: 7, NUMBER OF VARIABLES: 6
```

Figure 5.1f.

Figure 5.1c shows the error message caused by a missing element name. As in Figure 5.1b this message is produced during the input step and follows the statement in error. Again the position at which the error was detected is indicated by a question mark below the statement.

Figure 5.1d shows the error message caused by the use of an undefined terminal typename in a TERMINALS statement. This message is produced during the input step and follows the last statement of the element definition. The third line of this message indicates that the undefined terminal type STRUCT was specified for terminals 1-3.

Figure 5.1e shows the error message caused by an equation which has a missing operator (*). This message is produced during the expansion step and the first line indicates that the message refers to instance

RADIO.AMP.RESISTOR(3).

The third line of this message lists the equation in error and indicates the position at which the error was detected by a question mark. Note that this position is one item to the right of the position where the operator should appear. The pound sign (#) shown here is a character generated by MAP to mark the end of the equation.

Figure 5.1f shows the warning message caused by a difference in the number of equations and the number of variables in the model. This message is produced during the output step. The third line of this message lists the number of equations and the number of variables present.

The various warning and error messages produced by MAP are listed in appendix D in order of their message numbers. Appendix D also includes a brief commentary for each message describing its cause and necessary corrective action.

5.2. Model Dictionary

The purpose of the model analysis process is to translate the user's MODEL language model into an equivalent FORTRAN model, which can be interfaced to the numerical simulation process and controlled from the run-control program. The model analysis process includes an option to produce a _model dictionary_ which lists the corresondences between variables and parameters of the MODEL language model and the associated FORTRAN array elements in which these variables and parameters are maintained during the numerical simulation. This information must be available for checking or modifying the FORTRAN model and can be obtained by including a MAP statement, whose general form is simply

MAP

before the first element definition of the input file.

The model dictionary consists of two parts; a _node map_ (section 5.3), and a _variable map_ (section 5.4). These are shown for the models in appendix B. The node map and the variable map list each element instance of the model. Each instance is identified by an _invocation list_ and an _instance number_. The invocation list traces the invocation of the instance back to the highest level element of the model. For example, the invocation list for the left-most instance of element RAIN in the storm sewer model of appendix B is written

SEWER.RAIN

because this instance of element RAIN is invoked by element SEWER (the highest level element of the model). If element RAIN invoked a subelement named X, the invocation list for the instance of X invoked by this instance of RAIN would be written

Since the same subelement may be invoked more than once in the same element, an occurrence number is included in the invocation list to uniquely identify each instance in a model. This number appears in parentheses at the end of the invocation list. The occurrence numbers are assigned as the instances are processed during the model analysis. For each particular element, the first instance of that element is assigned occurence number 1, the second is assigned occurrence number 2, etc. Occurrence numbers are unique for all instances of a given element, but they are not unique for all instances in a model. To obtain a unique numerical designation for each instance, consecutive instance numbers are also assigned as the instances are processed.

The order in which instances are processed is determined by the model structure. The first instance to be processed is the highest level element of the model. This is instance 1. Instance 2 is the first subelement (if any) of instance 1 to be invoked in the definition of instance 1. Instances are numbered in this fashion until an instance of a primitive element is reached. After the primitive instance has been assigned an instance number, the next instance is the next subelement (if any) of the instance which invoked the primitive instance and the assignment of instance numbers proceeds as described above. If the instance which invoked the primitive instance has no more subelements, it is treated as if it were a primitive instance and the assignment of instance numbers again proceeds as described above. This continues until there are no more instances to be assigned instance numbers.

The general rule which determines the order of instance numbers can be summarized as follows. Instance 1 is the highest level element of the model. After instance N has been numbered, instance N+1 will be the first of the alternatives listed below which can be applied:

1. the first subelement (if any) of instance N;
2. the next subelement (if any) of the instance which invoked instance N;
3. the next subelement (if any) of the instance which invoked the instance which invoked instance N;
4. and so on.

The instance number of each instance is listed in the variable map, where it appears in parentheses as a part of the invocation list. The occurrence number is listed first, followed by a dash and the instance number. Thus, the instances used in the storm sewer model (see appendix B) are identified in the variable map as

```
ELEMENT SEWER(1-1)
ELEMENT SEWER.RAIN(1-2)
ELEMENT SEWER.RAIN(2-3)
ELEMENT SEWER.RESERV(1-4)
ELEMENT SEWER.RESERV(2-5)
ELEMENT SEWER.CAP(1-6)
ELEMENT SEWER.DRAIN(1-7)
ELEMENT SEWER.DRAIN(2-8)
ELEMENT SEWER.CAP(2-9)
ELEMENT SEWER.RESERV(3-10)
ELEMENT SEWER.CAP(3-11)
ELEMENT SEWER.SINK(1-12)
```

(SINK is a modified DRAIN which is used to drain the lower reservoir. It has only one terminal and simply accepts the applied flow without transmitting it, since the flow disappears from the model at this point.) In the node map, the invocation lists include only the occurrence numbers and not the instance numbers of each instance.

5.3. Node Map

The node map characterizes each node of each instance by listing the ID numbers of the connected terminals. For example, the node at which the left-most reservoir in the storm sewer model is connected to its drain is characterized in the node map for SEWER(1) as

```
NODE(3):  3, 19, 14,
```

Here the ID number 3 refers to local terminal 3 of SEWER(1) and the ID numbers 19 and 14 refer to the external terminals of the subelements DRAIN and RESERV which are connected through local terminal 3. The corresondence between the ID numbers 19 and 14 listed above and external terminals 0 of SEWER.DRAIN(1) and 2 of SEWER.RESERV(1) is shown in the node maps for those instances. For each external terminal of a node (typically there is only one) a different ID number is listed on the line following the list of connected terminals. This is the ID number used to represent the external terminal in the node map for the instance at the next higher level. These ID numbers permit the user to trace the external connections of an instance through the node map for the instance which invoked it. For example, the node consisting of external terminal 0 of SEWER.DRAIN(1) is characterized in the node map for SEWER.DRAIN(1) as

```
NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 19,
```

and the node consisting of external terminal 2 in SEWER.RESERV(1) is characterized in the node map for SEWER.RESERV(1) as

```
NODE(2): 2,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 14,
```

Thus, from looking at these three portions of the node map the user can determine that external terminal 0 of SEWER.DRAIN(1), external terminal

2 of SEWER.RESERV(1), and local terminal 3 of SEWER(1) form a node.

This node information is very useful in tracing terminal variables of the model to the corresponding FORTRAN variables of the numerical simulation. The manner in which the node map is used to find these correspondences is described in the following paragraphs.

Because E-set variables on one terminal are equivalent to the corresponding E-set variables on connected terminals, only one FORTRAN variable is assigned for each E-set member of any node. Therefore, the variable map will list E-set members and their corresponding FORTRAN variables at only one terminal of each node. The node map indicates the possible locations of this listing. For example, the node map for the storm sewer model indicates that the FORTRAN variable which corresponds to the Bernoulli quantity at the connection between the left-most reservoir and its drain will be listed in one (and only one) of the following places in the variable map:

i) with B(3) under SEWER(1)

ii) with B(0) under SEWER.DRAIN(1)

iii) with B(2) under SEWER.RESERV(1)

(In fact, it is listed with B(2) under SEWER.RESERV(1).)

The corresponding FORTRAN variable for an I-set member at an external terminal of one instance is listed in the variable map for the instance at the next higher level. The node map indicates the ID number used to represent the external terminal in the instance at the next higher level. For example, the node map for the storm sewer model indicates that the FORTRAN variable which corresponds to the flow at external terminal 0 of SEWER.DRAIN(1) will be listed with FLOW(19) in the variable map for SEWER(1) and the FORTRAN variable which corresponds to the flow at external terminal 2 of SEWER.RESERV(1) will be listed with FLOW(14) in the variable map of SEWER(1).

5.4. Variable Map

The variable map is listed after the node map and is subdivided by instances in the same way. The variable map for each instance is further subdivided under the headings GLOBAL VARIABLES, LOCAL VARIABLES, PARAMETERS and TERMINAL VARIABLES. Under each heading are listed the MODEL language names and the corresponding FORTRAN names for the variables or parameters. (These headings are omitted when there are no entries under them.) The FORTRAN names are either constants or elements of one of the FORTRAN vectors ZG, ZT, ZL or ZY. Any variable which appears in no equations will have NONE listed as its FORTRAN name. Any variable or parameter which is equivalenced to a constant by an equation or parameter assignment (e.g., G = 32, AS = 100.) will have that constant listed as its FORTRAN name. The remaining variables and parameters are partitioned into the four FORTRAN vectors named above. The space within these vectors is allocated sequentially--one element for a scalar, N*M elements for an N x M array. Only the necessary number of elements are declared in the FORTRAN model.

The FORTRAN names for two MODEL variables or parameters will be the same if those variables or parameters are equivalenced by an equation or parameter assignment (e.g., B(1) = HS). The FORTRAN names for two MODEL variables or parameters will be the same except for a leading minus sign on one of them if they are summed to zero by an equation or parameter assignment (e.g., FLOW(0)+FLOW(1) = 0).

In some cases MODEL will generate auxiliary variables and equations. These variables are listed at the end of the variable map under the heading

GENERATED VARIABLES

and their MODEL names are shown as either TPRIME or SUB. For the most part these variables are of nc concern to the user. However, there is one case of interest. For every reference to the square root function, SQRT(<u>arg</u>), which appears in a MODEL expression, a new variable, SUB, is generated. SUB then replaces the reference to SQRT in the FORTRAN model and the equation

SUB*SUB = <u>arg</u>

is added to the set of equations. This substitution is performed to avoid the possibility of an execution error arising if <u>arg</u> is temporarily negative during the computation of the initial state. A similar substitution is performed for each reference to the logarithm function, LOG, for the same reason. As a consequence of this substitution, the result of a reference to SQRT is not guaranteed to be positive unless it is forced positive by equations of the model. This is the reason that the absolute value function ABS is applied to the SQRT references in the equations of elements DRAIN and SINK in the storm sewer model in appendix B.

5.5. <u>Numerical Simulation Diagnostics</u>

The numerical simulation process produces diagnostics for numerical difficulties which arise during the simulation. These diagnostics concern severe numerical problems which cause NSP to terminate the steady state or transient analysis. In these cases an error message and the current values of all variables are printed and control is returned to the run control program. The termination code returned in the control variable ISTOP is the negative of the error code which caused the analysis to be discontinued. These error codes and the numerical difficulties from which they arise are listed below in Table 5.1.

Table 5.1. Numerical Simulation Error Codes

error code	corresponding numerical difficulty
1	ISTOP not reset after previous error.
2	NSP cannot compute an initial state for the model.
3	The specified error tolerance, EPS, cannot be met with a step size of HMIN.
4	The system of equations is ill-conditioned or singular.
5	An integration step failed to converge with a step size of HMIN. The system of equations is inconsistent or contains a discontinuous variable.

REFERENCES

1. Runge, T. F., "A universal language for network simulation," IMACS Conference, Virginia Polytechnic Institute, (March 1977).

2. Runge, T. F., "A universal language for network simulation," Ph.D. thesis, Department of Computer Science Report UIUCDCS-R-77-856, University of Illinois at Urbana-Champaign (1977).

3. CDC continuous system simulation language III (CSSL-III) user's guide, Form 17304400 Revision A, Control Data Corporation, Sunnyvale, California (1971).

4. Branin, F. H., G. R. Hogsett, R. L. Lunde, and L. E. Kugel, "ECAP II--a new electronic circuit analysis program," IEEE Journal on Solid State Circuits SC-6 4 (1971), 146-156.

5. Brennan, R. D. and R. N. Linebarger, "A survey of digital simulation:digital analog simulators," Simulation 3 (6) (1964), 244-258.

6. Watterberg, P., "DRAGON user's manual," available through the office of C. W. Gear, Department of Computer Science, University of Illinois at Urbana-Champaign (1977).

7. Hennegan, N., "GIML reference manual," Department of Computer Science Report UIUCDCS-R-77-857, University of Illinois at Urbana-Champaign (1977).

8. Hennegan, N., "GIMLI and GIML: A machine and programming language for low cost interactive graphical systems," Department of Computer Science Report UIUCDCS-R-77-856, University of Illinois at Urbana-Champaign (1977).

9. Rubner-Petersen, T., "An efficient algorithm using backward time-scaled differences for solving stiff differential-algebraic systems," Institute of Circuit Theory and Telecommunication, Technical University of Denmark (1973).

10. Gear, C. W., "Simultaneous numerical solution of differential-algebraic equations," IEEE, trans CT-18, no. 1, pp 89-94, Jan. 1972.

APPENDIX A. SUMMARY OF MODEL STATEMENTS AND INPUT FILE ORDERING

This appendix presents the general form and describes the effect of each MODEL statement. The input file of the model analysis process consists of four sections which must appear in the order below. The placement of these four sections in the job control program which invokes MAP and NSP is shown in appendix C. For other machines a similar placement of these sections is required.

Section 1. MODEL Control Statements.

The statements in this section apply to all elements in the model and can appear in any order.

general form of statement	page
ANALYZE name	26
specifies name of highest level element of model.	
RENAME name	33
specifies name to be used in place of TIME for independent variable.	
TYPE tname [E([ename[(d1[,d2])];]*)] [I([iname[(d1,[,d2])];]*)]	61
defines terminal type tname with E-set consisting of the variables ename and I-set consisting of the variables iname. d1 and d2 specify the optional dimensions of the variables.	

general form of statement	page
TITLE tstring	46
specifies title for solution history	
FLIST	47
causes MAP to print full FORTRAN listing (all FORTRAN produced by MAP).	
SLIST	47
causes MAP to print short FORTRAN listing (run-control program and subroutines RECORD and REPORT only).	
FUNCTION name n	51
specifies name and number of arguments of user-defined FORTRAN function subprogram.	
SUBROUTINE name n [RVECTOR [d1]¦ CVECTOR [d1]¦ ARRAY [d1 d2]]	51
specifies name, number of arguments, and dimensions of result of user-defined FORTRAN subroutine. See section 2.15 for description of how to reference subroutines.	
MAP	104
causes MAP to produce model dictionary.	
ABORT	99
causes numerical simulation to be aborted if MAP produces any warning messages.	

Section 2. Element Definitions.

Elements can be defined in any order. Statements within an element definition can appear in any order with the exception that NON-DEFAULTS and SUB-CONNECT statements must follow the corresponding SUB-ELEMENT statement.

general form of statement	page
ELEMENT name [parm[(d1[,d2])][=defexp];]*	28
specifies name of element to be defined by statements which follow and defines parameters parm and their default values defexp.	
DEFAULTS [parm[(d1,[,d2])][=defexp];]*	29
defines parameters parm and their default values defexp.	
LOCALS [var[(d1[,d2])];]*	31
defines local variables var.	
GLOBALS [var[(d1[,d2])];]*	32
defines global variables var.	
TERMINALS [id1[-id2] [type]]*	59
defines terminals numbered id1 through id2 with local scope and terminal type type.	
EXTERNALS [id1[-id2]]*	59
specifies external scope for terminals id1 through id2.	
SUB-ELEMENT name [parm=exp;]*	64
invokes element name as a subelement and specifies non-default values exp for its parameters parm.	

<u>general form of statement</u>	<u>page</u>
NON-DEFAULTS [<u>parm</u>=<u>exp</u>;]*	64
specifies non-default values <u>exp</u> for the parameters <u>parm</u> of the subelement invoked by the last previous SUB-ELEMENT statement.	
SUB-CONNECT [<u>id</u>]*	65
specifies the terminals <u>id</u> of this element to which the external terminals of the subelement invoked by the last previous SUB-ELEMENT statement are to be connected.	
CONNECT [<u>id</u>1-<u>id</u>2]*	67
specifies connection between terminal <u>id</u>1 and terminal <u>id</u>2 which are either local or external terminals of this element.	
EQUATIONS [<u>lhs</u>=<u>rhs</u>;]*	33
specifies equations	
EQUATIONS IF <u>cond</u> THEN <u>lhs</u>=<u>rhs</u> [ELSE IF <u>cond</u> THEN <u>lhs</u>=<u>rhs</u>]* ELSE <u>lhs</u>=<u>rhs</u>	37
specifies conditional equation	
EQUATIONS IF <u>cond</u> THEN STOP[=<u>n</u>]	42
specifies a termination condition <u>cond</u> and an associated termination code <u>n</u>.	
INITIAL [<u>var</u>=<u>const</u>;]*	40
specifies initial values <u>const</u> for variables or derivatives <u>var</u>.	
OUTPUT [<u>var</u>;]*	44
specifies output variables <u>var</u>.	

Section 3. Table Function Definitions.

Table functions may be defined in any order. For each table function the data point-function value pairs must appear in increasing order of the data points.

general form of statement	page
TABLE *tnum*	48

```
TABLE tnum
  t1    x1
  t2    x2
   .     .
  tn    xn
```

defines an empirical data function which is referenced by the table number *tnum*. The data point-function value pairs *t*i *x*i must appear in increasing order of the *t*i.

Section 4. Run Control File.

The run control file is placed in a different part of the input deck than the model definition statements of sections 1-3, as shown in appendix C. If the run control program is supplied by the user, the $NORCP statement must be the first statement of this section and the $DECLARE statement must be included immediately before the first executable statement of the main program or subroutine where it appears. Any other FORTRAN routines which are passed through MAP must follow the run control program. If the default run control program is used, neither the $NORCP nor the $DECLARE statement should be included in this section and only executable FORTRAN statements may be used in the run control file.

APPENDIX B -- MODEL EXAMPLES

The MODEL examples below were run on a CDC Cyber 175. In the second example, subroutine RECORD was modified to print the results using a D16.6 format instead of the default D26.16 format. The format for the variable labels was similarly modified so that each report would fit on one line.

B.1 Projectile Example.

In section 1.1 it is shown that the motion of a projectile under the influence of a constant gravitational force can be described by the equations:

$$x' = s$$
$$s' = -g,$$

$$\text{where} \quad x(0) = 0, \quad s(0) = v.$$

The MODEL formulation and solution of these equations is shown below. A termination condition halts the simulation when the projectile strikes the earth. Note that although the results are printed to 16 digits in the default format, the requested accuracy, as set by the control variable EPS, is only 5 digits. The model dictionary and short FORTRAN listing are produced by the MAP and SLIST options, respectively.

```
***************** MODEL -- CYBER VERSION 1.0 (6/23/78) *****************


 MAP(PASS1) START
-------------------



TITLE PROJECTILE SIMULATION
  SLIST
  MAP
ELEMENT PROJECT G; V
  LOCALS X;S
  EQUATIONS X' = S; S' = -G
    IF X < 0 & S < 0 THEN STOP = 1
  INITIAL X = 0; S = V
  OUTPUT X; S
--------------------
 PASS1:DATA REGIONS
 AREA ONE:
 REGION USED                    123
 AREA TWO:
 REGION USED                     86
--------------------
 MAP(PASS2) START
 PASS2:DATA REGIONS
 AREA ONE:
 REGION USED                    158
 AREA TWO:
 REGION USED                    315
--------------------
 MAP(PASS3) START


 DICTIONARY - NODE MAP


 ELEMENT PROJECT(1)

 NO NODES


 DICTIONARY - VARIABLE MAP


 ELEMENT PROJECT(1-1)

 LOCAL VARIABLES     OUTPUT NAMES
X                     ZY(1)
S                     ZY(2)

 PARAMETERS          OUTPUT NAMES
G                     ZG(1)
V                     ZG(2)
STOP STR=1
```

```
--------------------
 MAP(PASS4) START


      PROGRAM RCP(INPUT,OUTPUT, TAPE5=INPUT,TAPE6=OUTPUT,TAPE1)
      IMPLICIT DOUBLE PRECISION (A-H,Q-Z)
      DOUBLE PRECISION ZY,ZD,ZL,ZT,ZG
     +,HMIN,HMAX,EPS,DEL,TSTART,TSTOP,COMINT
      COMMON/VARS/ZY(2),ZD(2),ZL(1),ZT(1),ZG(2)
      COMMON/SYSPRM/HMIN,HMAX,EPS,DEL,TSTART,TSTOP,COMINT,ISTOP
      COMMON /SWITCH/ISNAP,MODE,IREAD,IWRITE,ISAVE
      DOUBLE PRECISION TIME,X,S,G,V
      EQUIVALENCE (TIME,ZT(1)),(X,ZY(1)),(S,ZY(2)),(G,ZG(1)),(V,ZG(2))
      HMIN=1.D-6
      HMAX=1.D-2
      EPS=1.D-5
      DEL=1.D-5
      TSTART=0.D0
      TSTOP=1.D0
      COMINT=1.D-2
      ISTOP=0
      ISNAP=0
      MODE=0
      IREAD=5
      IWRITE=6
      ISAVE=1
C
C     RUN CONTROL FILE
C
C     CONTROL VARIABLES
        COMINT = .1
        TSTOP = 10.
C     RUN PARAMETERS
C       GRAVITATIONAL ACCELERATION (M/SEC**2)
        G = 10.
C       INITIAL VELOCITY (M/SEC)
        V = 5.
C
   10 CALL NSP
      END
      SUBROUTINE RECORD(INIT,IUNIT,N,A)
C RECORD WRITES THE N OUTPUT VALUES IN ARRAY A TO UNIT
C IUNIT WHEN INIT NOT EQUAL ZERO. WHEN INIT EQUAL ZERO
C RECORD WRITES THE TITLE AND LABELS
      DOUBLE PRECISION A(N)
      IF (INIT.NE.0) GOTO 9000
      WRITE(IUNIT,1)
    1 FORMAT(1X,29X,49H      PROJECTILE SIMULATION
     +)
      WRITE(IUNIT,2    )
2     FORMAT(1X,9X
     +,8HTIME    ,18X
     +,8HX       ,18X
     +,8HS       ,18X
     +)
      RETURN
 9000 WRITE(IUNIT,9100) A
 9100 FORMAT(1X,5D26.16)
      RETURN
      END
      SUBROUTINE REPORT(INIT,IUNIT,ZG,ZT,ZL,ZY,ZD)
C REPORT COPIES THE OUTPUT VARIABLES INTO ARRAY A
C AND PASSES A TO SUBROUTINE RECORD FOR OUTPUT
      DOUBLE PRECISION A(3),ZY,ZL,ZT,ZG,ZD
      DIMENSION ZY(2),ZD(2),ZL(1),ZT(1),ZG(2)
      A(1)=ZT(1)
      A(2)=ZY(1)
      A(3)=ZY(2)
      CALL RECORD(INIT,IUNIT,3,A)
      RETURN
      END


 END MAP(PASS4)
-------------------
```

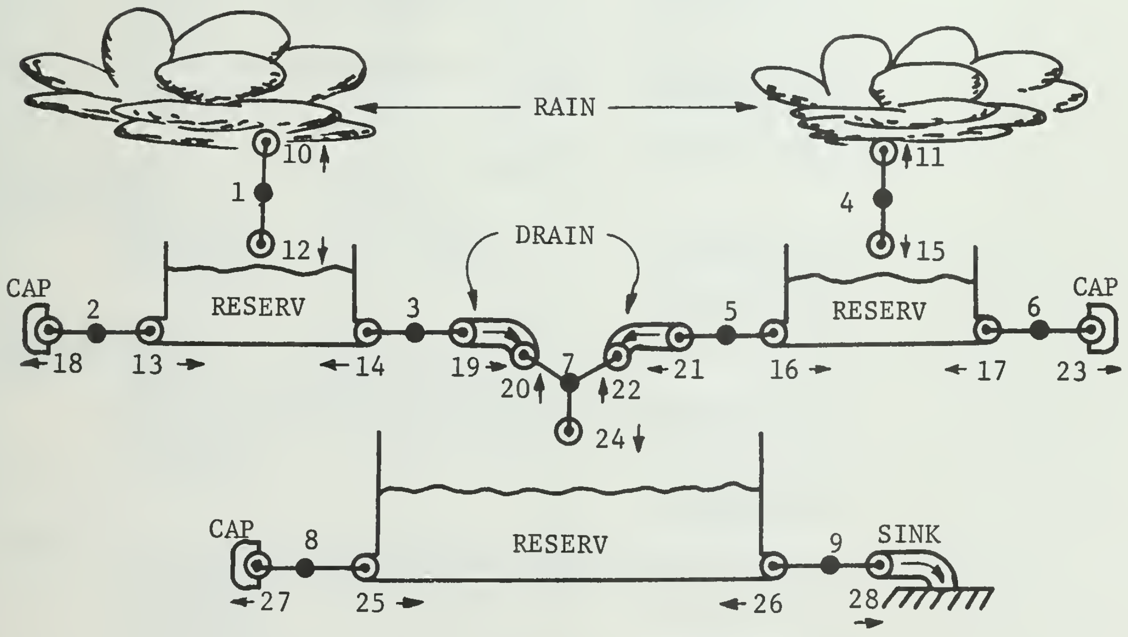
```
                                  PROJECTILE SIMULATION
        TIME                      X                         S
0.                        0.                        .5000000000000000D+01
 .1000000000000001D+00     .4499938677788106D+00    .3999999999999999D+01
 .2000000000000002D+00     .7999938677765877D+00    .2999999999999998D+01
 .3000000000000003D+00     .1049993867776588D+01    .1999999999999997D+01
 .4000000000000004D+00     .1199993867776588D+01    .9999999999999964D+00
 .5000000000000004D+00     .1249993867776587D+01   -.4440892098503464D-14
 .6000000000000005D+00     .1199993867776587D+01   -.1000000000000005D+01
 .7000000000000006D+00     .1049993867776586D+01   -.2000000000000006D+01
 .8000000000000007D+00     .7999938677765851D+00   -.3000000000000007D+01
 .9000000000000008D+00     .4499938677765840D+00   -.4000000000000008D+01
 .9999997667330982D+00   -.4965889175928534D-05   -.4999997667330982D+01

STOP CONDITION =  1
```

B.2 Sewer Network Example

This example shows the MODEL language formulation of a storm sewer network which is constructed from interconnected elements similar to the reservoir and drain elements described in section 3.8. Figure B.1 is a graphical representation of this network. Rainfall enters the system from the RAIN elements and is collected below the street in two small reservoirs. The water in these reservoirs drains into a large reservoir from which it drains out of the system (e.g., into a stream). The purpose of the simulation is to determine whether the reservoirs will overflow during a storm. This can be determined from the values of the variables which represent the surface heights of the reservoirs.

In addition to the formulation of this network, the model dictionary, run control program, and output subroutines are shown below.

Figure B.1. Storm Sewer Network.



```
***************** MODEL -- CYBER VERSION 1.0 (6/23/78) *****************


 MAP(PASS1) START
-------------------


%
%       STORM SEWER NETWORK
%
TITLE   STORM SEWER LEVELS
ANALYZE SEWER
        TYPE    FLUID E(B) I(FLOW)
        MAP
        SLIST
%
ELEMENT SEWER
        GAMMA=62.43; G=32.
        TERMINALS 1-9 FLUID
        GLOBALS GAMMA; G
%
                SUB-ELEMENT     RAIN    OAREA=1000
                SUB-CONNECT     1
%
                SUB-ELEMENT     RAIN    OAREA=1000
                SUB-CONNECT     4
%
                SUB-ELEMENT     RESERV  AS=100.;        HINIT=5.5
                                        A1=.75;         H1=5
                                        A2=.75;         H2=5
                SUB-CONNECT     1  2  3
%
                SUB-ELEMENT     RESERV  AS=100.;        HINIT=5.5
                                        A1=.75;         H1=5
                                        A2=.75;         H2=5
                SUB-CONNECT     4  5  6
%
                SUB-ELEMENT     CAP
                SUB-CONNECT     2
%
                SUB-ELEMENT     DRAIN   A=.75;          H=5
                SUB-CONNECT     3  7
%
                SUB-ELEMENT     DRAIN   A=.75;          H=5
                SUB-CONNECT     5  7
%
                SUB-ELEMENT     CAP
                SUB-CONNECT     6
%
                SUB-ELEMENT     RESERV  AS=1000. ;      HINIT=0.5
                                        A1=1;           H1=0.
                                        A2=1;           H2=0.
                SUB-CONNECT     7  8  9
%
```

```
                SUB-ELEMENT     CAP
                SUB-CONNECT     8
%
                SUB-ELEMENT     SINK    A=1;                H=0.
                SUB-CONNECT     9
%
ELEMENT RAIN    OAREA
        RAINRATE=(-.05*TIME*TIME+360*TIME)/7.776E7
        FLOW(0)=-RAINRATE*OAREA
        TERMINALS       0 FLUID
        EXTERNALS       0
        LOCALS  RAINRATE
%
ELEMENT RESERV  AS;  HINIT;  A1;  H1;  A2;  H2
        HS'*AS=FLOW(0)+FLOW(1)+FLOW(2)
        V1=FLOW(1)/A1
        V2=FLOW(2)/A2
        B(1)=V1*V1/(2*G)+P1/GAMMA+H1
        B(2)=V2*V2/(2*G)+P2/GAMMA+H2
        B(1)=HS
        B(2)=HS
        INITIAL         HS=HINIT
        OUTPUT          HS
        LOCALS  HS;  V1;  P1;  V2;  P2
        TERMINALS       0-2     FLUID
        EXTERNALS       0-2
%
ELEMENT CAP
        FLOW(0)=0
        TERMINALS       0  FLUID
        EXTERNALS       0
%
ELEMENT DRAIN   A;  H
        FLOW(0)+FLOW(1)=0
        FLOW(0)=ABS(SQRT(2*G*A*A*(B(0)-H)))
        TERMINALS       0-1  FLUID
        EXTERNALS       0-1
%
ELEMENT SINK    A;  H
        FLOW(0)=ABS(SQRT(2*G*A*A*(B(0)-H)))
        TERMINALS       0  FLUID
        EXTERNALS       0
--------------------
 PASS1:DATA REGIONS
 AREA ONE:
 REGION USED                1170
 AREA TWO:
 REGION USED                 466
--------------------
 MAP(PASS2) START
 PASS2:DATA REGIONS
 AREA ONE:
 REGION USED                1720
 AREA TWO:
```

```
REGION USED                 4376
--------------------
MAP(PASS3) START


DICTIONARY - NODE MAP


ELEMENT SEWER.RAIN(1)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 10,


ELEMENT SEWER.RAIN(2)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 11,


ELEMENT SEWER.RESERV(1)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 12,
NODE(1): 1,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 13,
NODE(2): 2,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 14,


ELEMENT SEWER.RESERV(2)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 15,
NODE(1): 1,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 16,
NODE(2): 2,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 17,


ELEMENT SEWER.CAP(1)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 18,


ELEMENT SEWER.DRAIN(1)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 19,
NODE(1): 1,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 20,
ELEMENT SEWER.DRAIN(2)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 21,
NODE(1): 1,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 22,


ELEMENT SEWER.CAP(2)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 23,


ELEMENT SEWER.RESERV(3)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 24,
NODE(1): 1,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 25,
NODE(2): 2,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 26,


ELEMENT SEWER.CAP(3)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 27,


ELEMENT SEWER.SINK(1)

NODE(0): 0,
IDENTICAL TERMINALS IN ELEMENT ABOVE: 28,


ELEMENT SEWER(1)

NODE(1): 1,12,10,
NODE(2): 2,18,13,
NODE(3): 3,19,14,
NODE(4): 4,15,11,
NODE(5): 5,21,16,
NODE(6): 6,23,17,
NODE(7): 7,24,22,20,
NODE(8): 8,27,25,
NODE(9): 9,28,26,


DICTIONARY - VARIABLE MAP


ELEMENT SEWER(1-1)
```

```
 GLOBAL VARIABLES    OUTPUT NAMES
GAMMA                 62.43D0
G                     32.D0

 TERMINAL VARIABLES  OUTPUT NAMES
FLOW(10)              ZT(3)
FLOW(11)              ZT(5)
FLOW(12)              -ZT(3)
FLOW(13)              -0
FLOW(14)              ZL(1)
FLOW(15)              -ZT(5)
FLOW(16)              ZL(4)
FLOW(17)              -0
FLOW(18)              0
FLOW(19)              -ZL(1)
FLOW(20)              ZL(1)
FLOW(21)              -ZL(4)
FLOW(22)              ZL(4)
FLOW(23)              0
FLOW(24)              ZL(7)
FLOW(25)              -0
FLOW(26)              ZL(8)
FLOW(27)              0
FLOW(28)              -ZL(8)


 ELEMENT SEWER.RAIN(1-2)

 LOCAL VARIABLES     OUTPUT NAMES
RAINRATE              ZT(2)

 PARAMETERS          OUTPUT NAMES
OAREA                 1000

 TERMINAL VARIABLES  OUTPUT NAMES
B(0)                  NONE


 ELEMENT SEWER.RAIN(2-3)

 LOCAL VARIABLES     OUTPUT NAMES
RAINRATE              ZT(4)

 PARAMETERS          OUTPUT NAMES
OAREA                 1000

 TERMINAL VARIABLES  OUTPUT NAMES
B(0)                  NONE


 ELEMENT SEWER.RESERV(1-4)

 LOCAL VARIABLES     OUTPUT NAMES
HS                    ZY(1)
```

```
V1                     ZG(1)
P1                     ZL(2)
V2                     ZY(2)
P2                     ZL(3)

 PARAMETERS           OUTPUT NAMES
AS                     100.D0
HINIT                  5.5D0
A1                     .75D0
H1                     5
A2                     .75D0
H2                     5

 TERMINAL VARIABLES   OUTPUT NAMES
B(1)                   ZY(1)
B(2)                   ZY(1)


 ELEMENT SEWER.RESERV(2-5)

 LOCAL VARIABLES      OUTPUT NAMES
HS                     ZY(3)
V1                     ZY(4)
P1                     ZL(5)
V2                     ZG(2)
P2                     ZL(6)

 PARAMETERS           OUTPUT NAMES
AS                     100.D0
HINIT                  5.5D0
A1                     .75D0
H1                     5
A2                     .75D0
H2                     5

 TERMINAL VARIABLES   OUTPUT NAMES
B(1)                   ZY(3)
B(2)                   ZY(3)


 ELEMENT SEWER.CAP(1-6)


 ELEMENT SEWER.DRAIN(1-7)

 PARAMETERS           OUTPUT NAMES
A                      .75D0
H                      5

 TERMINAL VARIABLES   OUTPUT NAMES
B(1)                   NONE


 ELEMENT SEWER.DRAIN(2-8)
 PARAMETERS           OUTPUT NAMES
A                      .75D0
H                      5


 ELEMENT SEWER.CAP(2-9)


 ELEMENT SEWER.RESERV(3-10)

 LOCAL VARIABLES      OUTPUT NAMES
HS                     ZY(7)
V1                     ZG(3)
P1                     ZL(9)
V2                     ZY(8)
P2                     ZL(10)

 PARAMETERS           OUTPUT NAMES
AS                     1000.D0
HINIT                  0.5D0
A1                     1
H1                     0.D0
A2                     1
H2                     0.D0

 TERMINAL VARIABLES   OUTPUT NAMES
B(1)                   ZY(7)
B(2)                   ZY(7)


 ELEMENT SEWER.CAP(3-11)


 ELEMENT SEWER.SINK(1-12)

 PARAMETERS           OUTPUT NAMES
A                      1
H                      0.D0


 GENERATED VARIABLES OUTPUT NAMES
SUB                    ZY(5)
SUB                    ZY(6)
SUB                    ZY(9)
--------------------
 MAP(PASS4) START


      PROGRAM RCP(INPUT,OUTPUT, TAPE5=INPUT,TAPE6=OUTPUT,TAPE1)
      IMPLICIT DOUBLE PRECISION (A-H,Q-Z)
      DOUBLE PRECISION ZY,ZD,ZL,ZT,ZG
     +,HMIN,HMAX,EPS,DEL,TSTART,TSTOP,COMINT
      COMMON/VARS/ZY(9),ZD(9),ZL(10),ZT(5),ZG(3)
      COMMON/SYSPRM/HMIN,HMAX,EPS,DEL,TSTART,TSTOP,COMINT,ISTOP
      COMMON /SWITCH/ISNAP,MODE,IREAD,IWRITE,ISAVE
      DOUBLE PRECISION TIME
      EQUIVALENCE (TIME,ZT(1))
      HMIN=1.D-6
      HMAX=1.D-2
      EPS=1.D-5
      DEL=1.D-5
      TSTART=0.D0
      TSTOP=1.D0
      COMINT=1.D-2
      ISTOP=0
      ISNAP=0
      MODE=0
      IREAD=5
      IWRITE=6
      ISAVE=1
C
C     RUN CONTROL FILE
C
C       CONTROL VARIABLES
        TSTOP = 7200.
        HMAX = 300.
        COMINT = 300.
        DEL = 1D-9
        EPS = 1D-4
C
   10 CALL NSP
      END
      SUBROUTINE RECORD(INIT,IUNIT,N,A)
C RECORD WRITES THE N OUTPUT VALUES IN ARRAY A TO UNIT
C IUNIT WHEN INIT NOT EQUAL ZERO. WHEN INIT EQUAL ZERO
C RECORD WRITES THE TITLE AND LABELS
      DOUBLE PRECISION A(N)
      IF (INIT.NE.0) GOTO 9000
      WRITE(IUNIT,1)
    1 FORMAT(1X,29X,49H        STORM SEWER LEVELS
     +,23H
     +)
      WRITE(IUNIT,2    )
2     FORMAT(1X,9X
     +,8HTIME    ,8X
     +,8HHS      ,8X
     +,8HHS      ,8X
     +,8HHS      ,8X
     +)
      RETURN
 9000 WRITE(IUNIT,9100) A
 9100 FORMAT(1X,5D16.6)
      RETURN
      END
      SUBROUTINE REPORT(INIT,IUNIT,ZG,ZT,ZL,ZY,ZD)
C REPORT COPIES THE OUTPUT VARIABLES INTO ARRAY A
C AND PASSES A TO SUBROUTINE RECORD FOR OUTPUT
```

```
      DOUBLE PRECISION A(4),ZY,ZL,ZT,ZG,ZD
      DIMENSION ZY(9),ZD(9),ZL(10),ZT(5),ZG(3)
      A(1)=ZT(1)
      A(2)=ZY(1)
      A(3)=ZY(3)
      A(4)=ZY(7)
      CALL RECORD(INIT,IUNIT,4,A)
      RETURN
      END

 END MAP(PASS4)
-------------------
```

```
                                           STORM SEWER LEVELS
         TIME                 HS                HS                HS
     0.                  .550000D+01       .550000D+01       .500000D+00
      .300000D+03        .504702D+01       .504702D+01       .767026D-01
      .600000D+03        .517274D+01       .517274D+01       .269104D+00
      .900000D+03        .535554D+01       .535554D+01       .566759D+00
      .120000D+04        .557560D+01       .557560D+01       .940511D+00
      .150000D+04        .581496D+01       .581496D+01       .136649D+01
      .180000D+04        .605755D+01       .605755D+01       .182185D+01
      .210000D+04        .628921D+01       .628921D+01       .228487D+01
      .240000D+04        .649773D+01       .649773D+01       .273481D+01
      .270000D+04        .667297D+01       .667297D+01       .315232D+01
      .300000D+04        .680674D+01       .680674D+01       .351952D+01
      .330000D+04        .689298D+01       .689298D+01       .382009D+01
      .360000D+04        .692774D+01       .692774D+01       .403959D+01
      .390000D+04        .690924D+01       .690924D+01       .416569D+01
      .420000D+04        .683792D+01       .683792D+01       .418857D+01
      .450000D+04        .671647D+01       .671647D+01       .410127D+01
      .480000D+04        .654990D+01       .654990D+01       .390030D+01
      .510000D+04        .634558D+01       .634558D+01       .358632D+01
      .540000D+04        .611330D+01       .611330D+01       .316515D+01
      .570000D+04        .586534D+01       .586534D+01       .264907D+01
      .600000D+04        .561646D+01       .561646D+01       .205902D+01
      .630000D+04        .538406D+01       .538406D+01       .142786D+01
      .660000D+04        .518821D+01       .518821D+01       .806299D+00
      .690000D+04        .505167D+01       .505167D+01       .276267D+00
      .720000D+04        .500000D+01       .500000D+01       .942437D-05

 STOP CONDITION =  0
```

APPENDIX C -- DECK STRUCTURE

This appendix lists and describes IBM 360/75 and CDC Cyber 175 job control statements to perform a MODEL analysis, FORTRAN compilation and numerical simulation as a sequence of steps in a single batch mode job. The job control included here illustrates the steps to be executed, the communication necessary between these steps (i.e., file I/O), and the various input streams for MODEL code, FORTRAN code, and data cards. It is not necessary to adhere to the job control given here to use the MODEL package. For example, the package can be used on other computers, and tape files can be used instead of disk files. It is also possible to perform the steps as a sequence of jobs. In this case, the job control shown here would be split into two or more parts (e.g., one job which performs the MODEL analysis, a second which performs the FORTRAN compilation, and a third which executes the numerical simulation). When this is done, the intermediate compiled codes must be saved on permanent files instead of the temporary ones used here.

C.1 CDC Cyber 175 Control Statements.

The MODEL analysis is a four step process. Binary files for the four steps are catalogued under the names P1BIN, P2BIN, P3BIN, and P4BIN. MAP uses the files in the first two rewind statements as I/O files for various purposes. ERFILE contains the fixed portions of the MAP error messages. SYSIN is used as the input file for the MODEL definition (sections 1, 2, and 3 of the input file as described in Appendix A) and RUNCF is used as the input file for the run control file (section 4 of the input file). These portions of the input files are placed following the control statements, separated by end-of-record markers as shown.

The MODEL source listing and any error messages are produced on PRINTF. The remaining files are all used for communication between steps. DATSTR and GLOB should be rewound between each of the MODEL analysis steps. The other files are rewound as necessary by MAP. PRINTF, SYSIN, and RUNCF and included as parameters to the loader calls for the four MAP steps. This allows the entire file of control statements to be defined as a control procedure and any desired file names can be substituted for these files when the procedure is invoked.

FIRFOR, EVLOUT, CPOUT, MXPOUT, RPOUT, and REPRT are used as intermediate files during the construction of the FORTRAN output, which appears in final form on MANOUT and OUTFOR. MANOUT contains the run control program, any other FORTRAN passed through MAP and subroutine NSP. OUTFOR contains the other FORTRAN subroutines produced by MAP. MAPFIL contains the model dictionary. The dictionary and the FORTRAN codes are copied to PRINTF if requested by the user.

The FORTRAN output files are packed into MANOUT and the FORTRAN compiler is then invoked to produce the binary file LGO. User-supplied FORTRAN may also be compiled at this point and included on LGO. LGO is then loaded with the binary of the numerical software, from the catalogued file NSPBIN, and the simulation is executed. The simulation produces printed output on file OUTPUT and a copy of the output is also produced on TAPE1. Data cards may be read by the run control program from the input deck if they are inserted in the indicated place between record separators.

```
jobname,cm120000,t30.
signon,<user-id>.
<password>.
bill,<dept,ps#>.
COPYCR,INPUT,SYSIN.
COPYCR,INPUT,RUNCF.
RFL,120000.
GET,ERFILE,P1BIN,P2BIN,P3BIN,P4BIN/UN=3RCX6EV.
REWIND,GLOB,DATSTR,FIRFOR,MAPFIL,PRINTF.
REWIND,MANOUT,OUTFOR,EVLOUT,CPOUT,MXPOUT.
REWIND,SYSIN,RUNCF,ERFILE,REPRT,RPOUT.
P1BIN,SYSIN,,PRINTF.
REWIND,GLOB,DATSTR.
P2BIN,,,,,,PRINTF.
REWIND,GLOB,DATSTR.
P3BIN,PRINTF,,RUNCF.
REWIND,GLOB,DATSTR.
P4BIN,PRINTF,,SYSIN.
REWIND,OUTFOR.
SKIPF,MANOUT.
COPY,OUTFOR,MANOUT.
PACK,MANOUT.
REWIND,MANOUT,LGO.
FTN,I=MANOUT,L=0.
FTN,L=0.
GET,NSPBIN.
REWIND,TAPE1.
LOAD,LGO.
NSPBIN.
7-8-9   (end-of-record)

        <MODEL Definition>

7-8-9   (end-of-record)

        <Run Control File>

7-8-9   (end-of-record)

        <User-supplied FORTRAN>

7-8-9   (end-of-record)

        <User Data Cards>

6-7-8-9 (end-of-file)
```

C.2 IBM 360/75 JCL.

The job control shown here uses two in-line procedures, WGO and WFORT. WGO invokes the IBM loader to link-edit, load, and execute one or more object decks from the file(s) associated with SYSLIN. WGO is used to execute the MODEL analysis and the numerical simulation. WFORT invokes the FORTRAN G compiler on the file(s) associated with SYSIN and writes the object deck for these codes to the disk file &&LINKFILE. It is used to compile the output from the MODEL analysis.

The MODEL analysis is a four step process. Object decks for the four steps are catalogued in the data set USER.P3503.LMOD under the names MP1, MP2, MP3, and MP4 and are executed by WGO. Object files IO$ and PLFSUBSX are also included from the same catalogued data set. IO$ contains the I/O support routines and PLFSUBSX contains the character manipulation routines used by MAP.

MAP uses I/O files declared as FORTRAN logical units 5, 6, and 91 through 99. Unit 5 is used for input streams. The MODEL language input (sections 1, 2, and 3 of the input file as described in Appendix A) follows the first step of the MODEL analysis. FORTRAN code which is passed through MAP (section 4 of the input file) follows the third step of the MODEL analysis. The model listing and any error messages which are produced are written to unit 6. The fixed portions of the MAP error messages are catalogued as USER.P3503.MACROS(ERMW) and are read as needed from unit 93. Temporary disk files are used for the remaining units. Units 91 and 92 are used for communication between steps. Units 97, 98, and 99 are used as intermediate files during the construction of the FORTRAN output, which is written in final form on units 95 and 96.

Unit 95 contains the run control program, any other FORTRAN passed through MAP, and subroutine NSP. Unit 96 contains the other FORTRAN subroutines produced by MAP. The model dictionary is produced on unit 94. The dictionary and the FORTRAN files are copied to unit 6 if requested by the user.

The FORTRAN output of the MODEL analysis is compiled by WFORT. The input file, SYSIN, consists of the disk files &&BLK95 and &&BLK96 created by MAP on FORTRAN units 95 and 96. User-supplied FORTRAN can also be included in this step. The object code produced by the FORTRAN compiler is written to the temporary disk file &&LINKFILE.

The object deck representing the model, &&LINKFILE, is linked and loaded with the object deck of the numerical simulation software, which is catalogued in USER.P3503.LMOD(NSP), in the final job step. FORTRAN unit 5 is used for user-supplied data cards and unit 90 is used for temporary disk storage. The output from the simulation is written to the output unit 6.

```
//MODEL  JOB
/*ID PS=<ps#>
/*ID CODE=<code>
/*ID NAME='<name>'
/*ID LINES=5000
/*ID IOREQ=5000
/*ID 360=45
/*ID REGION=250K
***
***   PROCEDURE DEFINITIONS
***
//WGO   PROC    LIBFIL2='USER.P3503.LMOD',PROG=IBMLOADR,
//              LIBFILE='SYS1.NULLIB',OBJDISP=PASS
//GO    EXEC    PGM=&PROG,REGION=56K,PARM=LET
//FT06F001  DD  SYSOUT=A
//SYSLIB    DD  DSN=&LIBFIL2,DISP=SHR
//          DD  DSN=FORTUOI,DISP=SHR
//          DD  DSN=&LIBFILE,DISP=SHR
//SYSLIN    DD  DSN=&&LNKFILE,UNIT=DISK,
//              SPACE=(TRK,1),DISP=(MOD,&OBJDISP)
//          DD  DDNAME=SYSIN
//SYSLOUT   DD  SYSOUT=A
//SYSPRINT  DD  SYSOUT=A
//SYSUDUMP  DD  SYSOUT=A
//      PEND
***
//WFORT PROC    LEVEL=G
//FORT  EXEC    PGM=UIFORT&LEVEL,REGION=116K
//SYSLIN    DD  DSN=&&LNKFILE,DISP(MOD,PASS),
//              SPACE=(TRK,(20,20)),UNIT=DISK,DCB=BLKSIZE=400
//SYSPRINT  DD  SYSOUT=A
//SYSPUNCH  DD  DUMMY
//SYSTEMP   DD  DSN=&&SYSUT3,UNIT=DISK,SPACE=(800,50,,,ROUND),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSUT1    DD  DSN=&&SYSUT1,UNIT=DISK,SPACE=(1050,50,,,ROUND)
//SYSUT2    DD  DSN=&&SYSUT2,UNIT=DISK,SPACE=(4096,30,,,ROUND)
//      PEND
***
***   4 PASS MODEL COMPILATION
***
//P1 EXEC WGO,PARM='LET,MAP,EP=PLWMAIN',REGION=250K
//SYSLIN DD DSN=USER.P3503.LMOD(P1),DISP=SHR
// DD DSN=USER.P3503.LMOD(IO$),DISP=SHR
// DD DSN=USER.P3503.LMOD(PLFSUBSX),DISP=SHR
//SYSUDUMP DD DUMMY
//FT93F001 DD DSN=USER.P3503.MACROS(ERMW),LABEL=(,,,IN),DISP=SHR,
// DCB=BUFNO=1
//FT96F001 DD DSNAME=&&BLK96,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(NEW,PASS)
//FT91F001 DD DSNAME=&&BLK91,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=1230,BUFNO=1),
// DISP=(NEW,PASS)
//FT92F001 DD DSNAME=&&BLK92,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=3510,BUFNO=1),
// DISP=(NEW,PASS)
//FT05F001 DD *
***
***      <MODEL Language Input Statements>
***
//P2 EXEC WGO,PARM='LET,NOMAP,EP=PLWMAIN',REGION=250K
//SYSLIN DD DSN=USER.P3503.LMOD(P2),DISP=SHR
// DD DSN=USER.P3503.LMOD(IO$),DISP=SHR
// DD DSN=USER.P3503.LMOD(PLFSUBSX),DISP=SHR
//SYSUDUMP DD DUMMY
//FT93F001 DD DSN=USER.P3503.MACROS(ERMW),LABEL=(,,,IN),DISP=SHR,
// DCB=BUFNO=1
//FT94F001 DD DSNAME=&&BLK94,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600,BUFNO=1),DISP=(NEW,PASS)
//FT91F001 DD DSNAME=&&BLK91,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=1230,BUFNO=1),
// DISP=(OLD,PASS)
//FT92F001 DD DSNAME=&&BLK92,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=3510,BUFNO=1),
// DISP=(OLD,PASS)
//P3 EXEC WGO,PARM='LET,NOMAP,EP=PLWMAIN',REGION=250K
//SYSLIN DD DSN=USER.P3503.LMOD(P3),DISP=SHR
// DD DSN=USER.P3503.LMOD(IO$),DISP=SHR
// DD DSN=USER.P3503.LMOD(PLFSUBSX),DISP=SHR
//SYSUDUMP DD DUMMY
//FT93F001 DD DSN=USER.P3503.MACROS(ERMW),LABEL=(,,,IN),DISP=SHR,
// DCB=BUFNO=1
//FT94F001 DD DSNAME=&&BLK94,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600,BUFNO=1),DISP=(MOD,PASS)
//FT95F001 DD DSNAME=&&BLK95,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(NEW,PASS)
//FT96F001 DD DSNAME=&&BLK96,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(MOD,PASS)
//FT97F001 DD DSNAME=&&BLK97,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(NEW,DELETE)
//FT98F001 DD DSNAME=&&BLK98,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(NEW,DELETE)
//FT91F001 DD DSNAME=&&BLK91,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=1230,BUFNO=1),
// DISP=(OLD,PASS)
//FT92F001 DD DSNAME=&&BLK92,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=3510,BUFNO=1),
// DISP=(OLD,PASS)
//FT05F001 DD *
***
***       <Run Control File>
***
//P4 EXEC WGO,PARM='LET,NOMAP,EP=PLWMAIN',REGION=250K
//SYSLIN DD DSN=USER.P3503.LMOD(P4),DISP=SHR
// DD DSN=USER.P3503.LMOD(IO$),DISP=SHR
// DD DSN=USER.P3503.LMOD(PLFSUBSX),DISP=SHR
//SYSUDUMP DD DUMMY
//FT95F001 DD DSNAME=&&BLK95,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(MOD,PASS)
//FT96F001 DD DSNAME=&&BLK96,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(MOD,PASS)
//FT99F001 DD DSNAME=&&BLK99,UNIT=SYSDA,SPACE=(TRK,(10,2)),
// DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),DISP=(NEW,DELETE)
//FT91F001 DD DSNAME=&&BLK91,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=1230,BUFNO=1),
// DISP=(OLD,PASS)
//FT92F001 DD DSNAME=&&BLK92,UNIT=SYSDA,
// SPACE=(TRK,(10,2)),DCB=(RECFM=VS,BLKSIZE=3510,BUFNO=1),
// DISP=(OLD,PASS)
***
***    FORTRAN COMPILATION
***
//       EXEC    WFORT,PARM='NOSOURCE,NOMAP'
//SYSIN     DD   DSNAME=&&BLK95,UNIT=SYSDA,SPACE=(TRK,(10,2)),
//               DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),
                 DISP=(OLD,DELETE)
//          DD   DSNAME=&&BLK96,UNIT=SYSDA,SPACE=(TRK,(10,2)),
//               DCB=(RECFM=FB,LRECL=72,BLKSIZE=1440,BUFNO=1),
//              DISP=(OLD,DELETE)
//          DD  *
***
***             <User-supplied FORTRAN>
***
***
***   NUMERICAL SIMULATION
***
//      EXEC    WGO,PARM='LET,EP=MAIN',REGION=150K
//SYSLIN    DD  DSN=&&LNKFILE,UNIT=DISK,DISP=(OLD,DELETE)
//          DD  DSN=USER.P3503.LMOD(NSP),DISP=SHR
//FT90F001  DD  DSN=USER.P3503.TEMP,UNIT=DISK,VOL=SER=UIPUB1,
//              SPACE=(1024,(25,10),RLSE),DISP=(NEW,DELETE),
//              DCB=(RECFM=VBS,LRECL=1020,BLKSIZE=1024)
//FT05F001  DD  *
***
***      <User-supplied Data Cards>
***
/*
```

APPENDIX D -- ERROR MESSAGES

D.1 MAP Error Messages

This section lists the MAP error and warning messages and describes the necessary corrective action for each. The diagnostic messages produced by the model analysis process are listed first. These messages are presented here in order of their message numbers. Warning messages are denoted by a 'w' appended to the message numbers. Only the second and third lines of these messages are shown here. The second line describes the error or possible error to which the message refers and the third line identifies the cause of the error or possible error. The first line of the message, which gives the message number, indicates whether it is a warning or an error message, and identifies the instance to which the message refers or the step of MAP during which it was produced, is not shown here. (See section 5.1 for a general description of the makeup of MAP diagnostic messages.)

Ordinary type is used below for the second line and for the fixed portions of the third line of MAP messages. Portions of the third line which are specific to a particular occurrence of the message are underlined. Messages which are listed by MAP immediately after the source statement in error do not have a third line. These messges are identified below by the line

(follows erroneous MODEL statement)

which follows the second line of the message. The third line is shown as equation

for messages which note the erroneous equation, parameter assignment, or initial value assignment and indicate the position at which the error was detected by a question mark. When an equation or assignment is listed in this way, the character # is used to mark its end.

MAP ERROR AND WARNING MESSAGES

1 MAP STORAGE OVERFLOW. ANALYSIS ABORTED.
PLW TERMINAL ERROR NUMBER n.

This message arises from a lack of sufficient storage space which causes the model anlaysis to be aborted. Occurrences of this message should be shown to the person responsible for maintaining MODEL at the user's installation.

2 not used

3 not used

4 ELEMENT IS A SUBELEMENT OF ITSELF. ANALYSIS ABORTED.

The definition of the element identified by the first line of this message invokes itself as a subelement. The element definition must be changed so that the element does not invoke itself.

5 not used

6w EQUATION IN MUTUALLY INDEPENDENT VARIABLES. MAY BE INCONSISTENCY.

An equation of the model invokes only run-time parameters, the independent variable, and variables whose values have already been determined by the values of the run-time parameter and the independent variable. The variables in this equation are overspecified and the equation is either an unnecessary identity or an inconsistency. In the latter case, the equation must be removed from the model.

7w SYSTEM MAY BE SINGULAR.
NUMBER OF EQUATIONS ≠ NUMBER OF VARIABLES.
NUMBER OF EQUATIONS: n, NUMBER OF VARIABLES: m.

If the number of equations is greater than the number of variables, the model is overspecified and extraneous equations should be removed from the model. If the number of equations is less than the number of variables, the model is underspecified and more equations may be needed. (See section 3.7.)

8 not used

9 not used

10w IMPROPER UNIT NUMBER. OUTPUT SENT TO UNIT 6.
(follows erroneous UNIT statement)

The UNIT statement does not specify the unit number as a positive integer. The solution history will be recorded on unit 6.

11w MODEL NAME SHORTENED TO 8 CHARACTERS.
(follows erroneous ANALYZE statement)

The ANALYZE statement specifies a string of more than 8 characters as the name of the highest level element. If the ANALYZE statement is not corrected, the first 8 characters of the specified name will be used.

12 TERMINAL TYPE DEFINITION IGNORED. TYPE NAME MISSING.
(follows erroneous TYPE statement)

The terminal type name and the names of its E-set and I-set variables must be specified in the TYPE statement.

13w TERMINAL TYPE NAME SHORTENED TO 8 CHARACTERS.
(follows erroneous TYPE statement)

The terminal type name specified in the TYPE statement is longer than 8 characters. If the TYPE statement is not corrected, the first 8 characters of the specified name will be used.

14 not used

15 not used

16 TERMINAL TYPE HAS NO VARIABLES.

The names of the terminal type's E-set and I-set variables must be specified in the TYPE statement.

17 FUNCTION DEFINITION IGNORED. TOO MANY FUNCTIONS DEFINED.
(follows erroneous FUNCTION or SUBROUTINE statement)

A maximum of 24 user-defined functions and subroutines may be used in the model. Additional function and subroutine syntax definitions are ignored.

18 FUNCTION DEFINITION IGNORED. FUNCTION NAME MISSING.
(follows erroneous FUNCTION or SUBROUTINE statement)

The name and number of arguments of the user-defined function or subroutine must be specified in the FUNCTION or SUBROUTINE statement.

19w FUNCTION NAME SHORTENED TO 8 CHARACTERS.

The name specified in the FUNCTION or SUBROUTINE statement is longer than 8 characters. If the statement is not corrected, the first 8 characters of the specified name will be used.

20 FUNCTION DEFINITION IGNORED. IMPROPER ARGUMENT COUNT.
(follows erroneous FUNCTION or SUBROUTINE statement)

The number of arguments used in referencing the user-defined function or subroutine must be specified as a positive integer in the FUNCTION or SUBROUTINE statement.

21 FUNCTION DEFINITION IGNORED. NO ARGUMENTS.
(follows erroneous FUNCTION or SUBROUTINE statement)

The number of arguments specified in the FUNCTION or SUBROUTINE statement is zero. Every user-defined function or subroutine must take at least one argument.

22 TABLE FUNCTION IGNORED. TABLE NUMBER NOT POSITIVE INTEGER.
(follows erroneous TABLE statement)

The table number must be specified as a positive integer in the TABLE statement.

23 TABLE FUNCTION DEFINITION IGNORED. TABLE NUMBER USED PREVIOUSLY.
(follows erroneous TABLE statement)

A unique table number must be specified for each table function defined.

24w INDEPENDENT VARIABLE NAME SHORTENED TO 8 CHARACTERS.
(follows erroneous RENAME statement)

The new name specified for the independent variable is longer than 8 characters. If the RENAME statement is not corrected, the first 8 characters of the specified name will be used.

25 IMPROPER DIMENSION. DIMENSION SET TO 1.
(follows erroneous SUBROUTINE statement)

A specified dimension of the result returned by the user-defined subroutine is not a positive integer. Dimension 1 will be used in place of the improper dimension.

26 ELEMENT NAME MISSING.
(follows erroneous ELEMENT statement)

The element name must be specified in the ELEMENT statement if the element is to be invoked as a subelement or in an ANALYZE statement.

27w ELEMENT NAME SHORTENED TO 8 CHARACTERS.

(follows erroneous ELEMENT statement)

The element name is longer than 8 characters. If the ELEMENT statement is not corrected, the first 8 characters of the specified name will be used.

28 UNDEFINED STATEMENT IGNORED.
(follows erroneous MODEL statement)

The first item of the preceding statement is not a legal MODEL keyword and the statement is ignored. The keyword should be corrected to appear as shown in appendix A.

29w SUBELEMENT NAME SHORTENED TO 8 CHARACTERS.
(follows erroneous SUBELEMENT statement)

The name specified in the SUBELEMENT statement is longer than 8 characters. If the SUBELEMENT statement is not corrected, the first 8 characters of the specified name will be used.

30 TERMINAL DECLARATION IGNORED. IMPROPER TERMINAL ID NUMBER(S).
TERMINAL ID NUMBER(S): _n_-_m_.

The terminal ID number(s) shown is not a non-negative integer. Terminals must be specified by unique non-negative integers in the TERMINALS statement.

31w TERMINAL ID NUMBERS SPECIFIED IN WRONG ORDER. ORDER REVERSED.
TERMINAL ID NUMBERS: _n_-_m_.

A consecutively numbered group of terminal ID's is specified in a TERMINALS statement as _n_-_m_ where _n_ is greater than _m_. If the TERMINALS statement is not corrected, the group of terminal ID's _m_-_n_ (_m_ through _n_) will be defined.

32w TERMINAL TYPE NAME SHORTENED TO 8 CHARACTERS.
TERMINAL TYPE NAME: _name_.

A terminal type name associated with a group of terminals in a TERMINALS statement is longer than 8 characters. If the TERMINALS statement is not corrected, the first 8 characters of the specified name will be used.

33 TERMINAL TYPE NOT ASSIGNED. TERMINAL TYPE NAME UNDEFINED.
TERMINAL ID NUMBERS: _n_-_m_, TERMINAL TYPE NAME: _name_.

The terminal type name _name_ has not been defined in a TYPE statement. A TYPE statement defining _name_ must be included or _name_ must be corrected to match a type name defined in a TYPE statement.

34 EXTERNAL TERMINAL DECLARATION IGNORED.
IMPROPER TERMINAL ID NUMBER(S), TERMINAL ID NUMBER(S): _n_-_m_.

The terminal ID number(s) shown is not a non-negative integer.

Terminals must be specified by non-negative integers in the EXTERNALS statement.

35 EXTERNAL TERMINAL DECLARATION IGNORED. TERMINAL NOT DEFINED.
TERMINAL ID NUMBER: n.

The terminal ID number shown has not been defiined in a TERMINALS statement. A terminal(s) which is declared as external in an EXTERNALS statement(s) must be defined in a TERMINALS statement(s) in the same element.

36 CONNECTION TO SUBELEMENT IGNORED. IMPROPER TERMINAL ID NUMBER(S).
TERMINAL ID NUMBER(S): n-m.

A terminal ID number or pseudo-ID number is not specified properly in a SUBCONNECT statement. The terminal ID number n must be a non-negative integer and the pseudo-ID number -m, if present, must be a negative integer.

37 CONNECTION TO SUBELEMENT IGNORED. TERMINAL NOT DEFINED.
TERMINAL ID NUMBER: n.

The terminal ID number shown has not been defined in a TERMINALS statement. A terminal which is specified as a connection to a subelement in a SUBCONNECT statement must be defined in a TERMINALS statement in the same element.

38 CONNECTION IGNORED. IMPROPER TERMINAL ID NUMBER(S).
TERMINAL ID NUMBER(S): n-m.

One of the terminal ID numbers shown is improperly specified in a CONNECT statement or one of the ID numbers is missing. The connection must be specified by two non-negative integer terminal ID numbers separated by -.

39 CONNECTION IGNORED. TERMINAL(S) NOT DEFINED.
TERMINAL ID NUMBERS: n-m.

One or both of the terminal ID numbers shown has not been defined in a TERMINALS statement. A terminal which is specified for connection in a CONNECT statement must be defined in a TERMINALS statement in the same element.

40 EQUATION IGNORED. BUILT IN FUNCTION ARGUMENT BEFORE ? NOT SCALAR.
equation

The MODEL built-in functions may only operate on scalar arguments. The argument of the built-in function referenced before ? must be changed to a scalar expression.

41 UNRECOGNIZABLE CHARACTER BEFORE ? IGNORED.
equation

The character preceding ? has no meaning in an equation and must be removed or replaced.

42 EQUATION IGNORED. OPERAND MISSING BEFORE ?.
<u>equation</u>

The equation shown contains two consecutive operators before ?. An operand, such as a parameter, variable, constant, or function reference, must be inserted between the operators or one of the operators must be removed.

43 EQUATION IGNORED. OPERATOR MISSING BEFORE ?.
<u>equation</u>

The equation shown contains two consecutive operands before ?. An operator must be inserted between the operands or one of the operands must be removed.

44w NAME SHORTENED TO 8 CHARACTERS BEFORE ?.
<u>equation</u>

The parameter, variable, function, or subroutine name before ? is longer than 8 characters. If the name is not corrected, the first 8 characters will be used.

45 EQUATION IGNORED. NUMERICAL STRING BEFORE ? IS TOO LONG.
<u>equation</u>

The numerical constant before ? exceeds the length limitation imposed by MAP. A numerical constant must contain no more than 24 characters (22 characters if exponential notation (e.g., 7.0E5) is not used).

46 EQUATION IGNORED. UNDEFINED VARIABLE NAME BEFORE ?.
<u>equation</u>

The parameter or variable name before ? must be defined within the specified element or as a global variable if it is to be referenced in an equation of this element.

47 EQUATION IGNORED. UNDEFINED TERMINAL ID NUMBER BEFORE ?.
<u>equation</u>

The terminal ID number before ? must be defined in a TERMINALS statement (or, if it is a pseudo-ID number, in a SUBCONNECT statement) of the specified element if it is to be referenced in an equation of this element.

48 EQUATION IGNORED. WRONG FORM FOR TERMINAL ID NUMBER BEFORE ?.
<u>equation</u>

The terminal ID number before ? must be an integer and must be followed by) when one of its terminal variables is referenced in an equation.

49 EQUATION IGNORED. REFERENCE TO UNTYPED TERMINAL BEFORE ?.
<u>equation</u>

The terminal whose ID number precedes ? or one of the terminals to which it is connected must be assigned a terminal type in a TERMINALS statement.

50 EQUATION IGNORED. UNDEFINED TERMINAL VARIABLE NAME BEFORE ?.
<u>equation</u>

The terminal variable reference before ? does not specify the name of a terminal variable associated with the terminal whose ID number is given. The user should check the definition of that terminal and of its associated terminal type.

51 EQUATION IGNORED. STRIP TERMINAL VARIABLE NOT IN FUNCTION.
<u>equation</u>

References to terminal variables of strip terminals are permitted only within arguments of user-defined functions or subroutines. The terminal variable before ? may not be referenced except in this fashion.

52 EQUATION IGNORED. TOO MAY FUNCTION ARGUMENTS BEFORE ?.
<u>equation</u>

The function or subroutine reference before ? contaiins too many arguments. Excess arguments must be removed. Built-in functions take a single argument. The number of arguments used in referencing a user-defined function or subroutine is specified in its FUNCTION or SUBROUTINE syntax definition statement.

53 EQUATION IGNORED. TOO FEW FUNCTION ARGUMENTS BEFORE ?.
<u>equation</u>

The function or subroutine reference before ? contains too few arguments. More arguments must be supplied. Built-in functions take one argument. The number of arguments used in referencing a user-defined function or subroutine is specified in its FUNCTION or SUBROUTINE syntax definition statement.

54 EQUATION IGNORED. INCOMPLETE FUNCTION REFERENCE BEFORE ?.
<u>equation</u>

A) and possibly more arguments are required to complete a function or subroutine reference before the equal sign or before the end of the equation.

55 EQUATION IGNORED. MISPLACED ' BEFORE ?.
<u>equation</u>

The differentiation operator ' before ? is not preceded by a parameter or variable name. The differentiation operator may only be applied to parameters and variables. It may not be applied to arbitrary expressions or to derivatives.

56 EQUATION IGNORED. UNMATCHED ((S) BEFORE ?.
<u>equation</u>

One or more ('s preceding ? do not have a matching). Parentheses must be used in matching pairs.

57 EQUATION IGNORED. UNMATCHED) BEFORE ?.
<u>equation</u>

The) before ? does not have a matching (. Parentheses must be used in matching pairs.

58 EQUATION IGNORED. MISPLACED , BEFORE ?.
<u>equation</u>

The comma before ? is not used to separate function or subroutine arguments and thus has no meaning in the equation. It must be removed.

59 EQUATION IGNORED. = MISSING BEFORE ?.
<u>equation</u>

The equation has no equal sign. An equal sign must be included.

60 EQUATION IGNORED. EXTRA = BEFORE ?.
<u>equation</u>

The equation contains a second equal sign. Only one equal sign may be included in an equation.

61 EQUATION IGNORED. IMPROPER FORM FOR PARAMETER ASSIGNMENT.
<u>equation</u>

The lefthand side of the parameter assignment shown must consist of a single parameter name. Other terms on the lefthand side must be removed.

62 EQUATION IGNORED. INTEGER DIMENSION EXPECTED BEFORE ?.
<u>equation</u>

The dimension specified for a variable or parameter before ? is not a positive integer. All dimensions must be positive integers.

63 EQUATION IGNORED. UNDEFINED PSEUDO TERMINAL ID NUMBER BEFORE ?.
<u>equation</u>

The pseudo-ID number before ? has not been defined in a SUBCONNECT statement. A pseudo-ID number must be defined in a SUBCONNECT statement of an element if it is to be referenced in an equation of that element.

64 IF-THEN-ELSE IGNORED. OPERATOR-OPERAND DISAGREEMENT BEFORE ?.
equation

The conditional equation shown has an improperly specified condition. Either a logical operator (&,¦) is applied to a numerical operand or a relational operator (<,>) is applied to a nonscalar or a logical operand (the result of a relational or a logical operator).

65 EQUATION IGNORED. ILLEGAL USE OF ¦, &, <, OR > INSIDE FUNCTION.
equation

The logical or relational operator before ? has no meaning within a function argument and must be removed.

66 IF-THEN-ELSE IGNORED. IF CLAUSE IS NOT A LOGICAL EXPRESSION.
equation

The IF clause of the conditional equation shown does not contain a relational operator and thus is not a logical (i.e., TRUE or FALSE) expression. An IF clause must specify a logical condition.

67 ELSE CLAUSE(S) IGNORED. NOT PRECEDED BY IF-THEN.
equation

The ELSE clause of the conditional equation shown is not preceded by IF and THEN clauses and thus is meaningless. IF and THEN clauses must be included before the ELSE clause.

68 IF-THEN-ELSE IGNORED. MISSING THEN CLAUSE.
equation

The ELSE clause of the conditional equation shown is not preceded by a THEN clause and thus is meaningless. A THEN clause must be included between the IF and ELSE clauses.

69 EQUATION IGNORED. UNMATCHED ((S) IN ARGUMENT BEFORE ?.
equation

The function or subroutine argument which is terminated by the comma before ? contains a (which has no matching). Parentheses must be used in matching pairs.

70 EQUATION IGNORED. UNDEFINED TABLE FUNCTION NUMBER BEFORE ?.
equation

The table function reference before ? refers to a table function which has not been defined in a TABLE statement. A table function must be defined and given a unique table number in a TABLE statement if it is to be used in equations of the model.

71 EQUATION IGNORED. COMMA MUST FOLLOW TABLE FUNCTION NUMBER.
equation

The table function reference before ? is not in the proper form. Table functions are referenced in the form TABLE (n, arg) where a comma separates the table number n from the argument arg.

72 INITIAL VALUE ASSIGNMENT IGNORED. IMPROPER FORM.
equation

The lefthand side of the initial value assignment shown must consist of a parameter, variable, or derivative. Other terms on the lefthand side must be removed.

73 EQUATION IGNORED. EXPRESSION UNDEFINED - IMPROPER DIMENSIONS.
equation

The =, -,+, *, or . operator before ? has operands whose dimensions do not agree. Both operands of =, -, and + must have the same dimensions. Both operands of . (vector product) must be vectors of the same length. The lefthand operand of * must be I x J and the righthand operand must be J x K for some I, J, and K.

74 EQUATION IGNORED. NO TERMINALS CONNECTED TO STRIP TERMINAL.
equation

The equation contains a reference to the terminal variables of a strip terminal before ? but no terminals have been connected to the strip terminal. The strip terminal must be connectd to at least one other terminal.

75w TERMINAL IS DOUBLY DEFINED.
TERMINAL ID NUMBER: n.

The terminal whose ID number is given has been defined twice in the TERMINALS statement(s)of this element. The model is unaffected by this repetition.

76w VARIABLE OR PARAMETER NAME SHORTENED TO 8 CHARACTERS.
VARIABLE OR PARAMETER NAME: name.

The variable or parameter name given is longer than 8 characters. If the variable or parameter declaration is not corrected, the first 8 characters of the name will be used.

77w DECLARATION OF INDEPENDENT VARIABLE IS NOT NECESSARY.

The independent variable has been declared in a GLOBALS or LOCALS statement. This is unnecessary but will not affect the model.

78 not used

79 not used

80 TERMINAL HAS NOT BEEN ASSIGNED A TYPE.
TERMINAL ID NUMBER: n.

The terminal whose ID number is given has not been assigned a type and neither has any terminal to which it is connected. The terminal must be assigned a type in the TERMINALS statement in which it is defined.

81 LOCAL CONNECTION IGNORED. TERMINAL TYPE MISMATCH.
TERMINAL ID NUMBERS: _n_, _m_, TERMINAL TYPES: _type1_, _type2_.

The connection of the two terminals whose ID numbers are given, which was specified in a CONNECT statement, is ignored because the terminals have different terminal types. Terminals of different types must not be connected.

82 EXTERNAL SUBELEMENT CONNECTION TERMINAL TYPE MISMATCH.
TERMINAL ID NUMBERS: _n_, _m_, TERMINAL TYPES: _type1_, _type2_.

The connection between external terminal _n_ and the terminal _m_ specified in the SUBCONNECT statement of the element at the next higher level is ignored because these terminals have different terminal types. Terminals of different types must not be connected.

83 EXTERNAL TERMINAL IN HIGHEST LEVEL ELEMENT REDECLARED AS LOCAL.
TERMINAL ID NUMBER: _n_.

Terminal _n_ is defined and declared external in the highest level element of the model. Since the highest level element is not a subelement of any other element, it must not have any external terminals.

84 EXTERNAL TERMINAL REDECLARED AS LOCAL. TOO FEW CONNECTIONS.
TERMINAL ID NUMBER: _n_.

No connection is specified for external terminal _n_ in the SUBCONNECT statement of the element at the next higher level. The SUBCONNECT statement(s) of the element at the next higher level must define a connection for each of the external terminals of the subelement.

85 EXTRA CONNECTIONS(S) TO ELEMENT IGNORED.

The SUBCONNECT statement(s) of the element at the next higher level specifies more connections to this element than the number of external terminals declared in this element. The model is unaffected by these extra connections, which are ignored.

86 REFERENCE TO UNDEFINED SUBELEMENT IGNORED.
SUBELEMENT NAME: _name_.

The element identified in the first line of this message invokes a subelement named _name_ but no element by that name is defined in the model. All subelements used in the model must be defined.

87w DERIVATIVE OF CONSTANT VARIABLE REPLACED BY 0.

VARIABLE NAME: _name_.

The derivative of the variable whose name is given appears in an equation but the variable is equivalenced to a constant by another equation. Thus its derivative is 0.

88w EQUIVALENCE OF TWO INDEPENDENT VARIABLES IGNORED.
VARIABLE NAMES: _name1_, _name2_.

An equation of the model equivalences the two variables whose names are given but each of these variables depends only on constants, run-time parameters, and the independent variable. Thus the equation is either an unnecessary identity or an inconsistency. This equation is ignored.

89w EQUIVALENCE OF INDEPENDENT VARIABLE AND CONSTANT IGNORED.
VARIABLE NAME: _name_.

An equation of the model equivalences the variable whose name is given to a constant but this variable has already been found to depend only on constants, run-time parameters, and the independent variable. Thus the equation is either an unnecessary identity or an inconsistency. This equation is ignored.

90 EXTERNAL CONNECTION HAS UNMATCHED STRIP TERMINAL.
TERMINAL ID NUMBER: _n_.

Either _n_ is a strip terminal and the SUBCONNECT statement in the element at the next higher level does not specify that _n_ is a strip terminal or _n_ is not a strip terminal and the SUBCONNECT statement specifies that _n_ is a strip terminal.

91 EXTERNAL STRIP TERMINAL NOT ASSIGNED A TYPE.
TERMINAL ID NUMBER: _n_.

External strip terminal _n_ has not been assigned a terminal type and neither have any of the terminals to which it is connected. A terminal type must be specified for terminal _n_ in the TERMINALS statement in which it is defined.

92 CONNECTION OF TWO STRIP TERMINALS IGNORED.
TERMINAL ID NUMBERS: _n_, _m_.

The two external strip terminals of subelements whose ID numbers are given are connected in a CONNECT statement but it is not meaningful to connect two strip terminals so this connection is ignored.

93 CONNECTION TO EXTERNAL STRIP TERMINAL IGNORED.
TERMINAL ID NUMBERS: _n_, _m_.

A connection between external strip terminal _m_ and terminal _n_ of the same element is specified in a CONNECT statement. It is not meaningful to connect a strip terminal to another terminal of the same element so this connection is ignored.

94 IF-THEN-ELSE IGNORED. ELSE CLAUSE MISSING AFTER
if-then clauses

The IF and THEN clauses shown are not followed by an ELSE clause and consequently they are ignored. An ELSE clause must be added.

95w INITIAL VALUE ASSIGNMENT FOR AN INDEPENDENT VARIABLE IGNORED.
VARIABLE INITIALIZED: *name*.

An initial value has been specified for the varible whose name is given but this variable has been found to depend only on constants, run-time parameters, and the independent variable. Thus this initial value is either an unnecessary identity or an inconsistency and consequently it is ignored.

96 INITIAL VALUE UNKNOWN AT TIME = TSTART. ASSIGNMENT IGNORED.
VARIABLE INITIALIZED: *name*.

The initial value specified for the variable or derivative whose name is given does not depend only on constants, run-time parameters, and the initial value of the independent variable. Thus the initial value assignment is ignored.

97w INITIALIZED VARIABLE APPEARS IN NO EQUATIONS.
VARIABLE INITIALIZED: *name*.

An initial value is specified for the variable or derivative whose name is given but this variable appears in no equations and consequently has no effect on the model.

98 IMPROPER DIMENSIONS. VARIABLE OR PARAMETER REDECLARED AS SCALAR.
VARIABLE OR PARAMETER DECLARATION: *name* (*d*1, *d*2).

The variable or parameter declaration given specifies a dimension which is not a positive integer. If the declaration is not corrected the variable or parameter will be treated as a scalar.

99 EQUATION IGNORED. CONTAINS MORE THAN 6 NESTED NONSCALAR FUNCTIONS.
equation

An equation of the model contains references to user-defined subroutines which return nonscalar results and which are nested to a depth greater than 6. A maximum of 6 levels of nesting is permitted for user-defined nonscalar subroutine references.

100w DECLARE WITHOUT NORCP CARD, OR EXTRA DECLARE CARD.

The $DECLARE and $NORCP keywords must both appear if either is used. Each keyword may be used only one time in the run control file.

D.2 NSP Error Messages

NSP error messages consist of an error number followed by a one-line message describing the nature of the error. All of the errors are fatal and cause the simulation to abort and control to return to the run control program with the negative of the error number in the control variable ISTOP. The first parenthesized name in the message identifies the subroutine which issued the error message. The second parenthesized name is the subroutine which detected the error. Possible causes and corrective actions are listed below for each of the NSP errors.

NSP ERROR MESSAGES

1 (P2) -- ERROR FLAG NOT RESET (P2).

The simulation model was called with ISTOP not equal to zero. This may occur during a multiple run simulation when a new run is started without checking the error flag from the preceding run. Either the preceding error condition must be corrected, or ISTOP must be reset before NSP is called.

2 (P2) -- INITIAL STATE NOT FOUND IN n ITERATIONS (DIFMF3).

This usually results from incorrect specification of the initial values for the model, which leads to an inconsistent system of equations for the unknown initial values. If no obvious error can be found in the model definition, the debug control variable may be set to ISNAP = 1 to inspect the iterations during the steady-state analysis. With the aid of the model dictionary, the initial values that were specified may be checked and troublesome variables or equations may be identified. If the error still cannot be located, it will be necessary to obtain a full listing of the FORTRAN model (using FLIST) in order to inspect the complete set of model equations. These are found in subroutine DIFFUN. The FORTRAN variable names can be translated to their MODEL names by means of the variable map obtained by using the MAP option. The dependence of the unknown initial conditions upon the specified initial conditions must then be checked by hand to determine why the initial state cannot be found.

3 (P2) -- INTEGRATION ERROR EXCEEDS EPS AT HMIN (DFASUB).

The specified error tolerance cannot be satisfied at the minimum stepsize. The problem may be cured by simply decreasing HMIN.

This error often indicates that the independent variable is poorly scaled. If HMIN is well below the estimated value given in section 4.3, the equations may be singular. When this error occurs at the start of the transient analysis, it may indicate that the steady-state solution was not unique and that some derivatives were arbitrarily assigned zero values. The remedy is to obtain a better steady state solution or decrease HMIN.

4 (P2) -- ERROR IN MATRIX INVERSION (DFASUB).

The system of equations has become inconsistent. This usually results from poor scaling and may require that the model equations be reformulated.

5 (P2) -- CORRECTOR DID NOT CONVERGE AT HMIN (DFASUB).

This error is most commonly caused by a discontinuity in the model equations. Such a problem may occur if a conditional equation causes a variable to be discontinuous. Functions and subroutines which are supplied by the user should also be checked for discontinuities. When this error occurs at the beginning of the transient analysis, the cause may be a bad initial state.

Form AEC-427
(6/68)
AECM 3201

U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIF C AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO. COO-2383-0047	2. TITLE SIMULATION OF CONTINUOUS NETWORKS WITH MODEL

3. TYPE OF DOCUMENT (Check one):

[✓] a. Scientific and technical report
[ ] b. Conference paper not to be published in a journal:
Title of conference
Date of conference
Exact location of conference
Sponsoring organization
[ ] c. Other (Specify)

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

[✓] a. AEC's normal announcement and distribution procedures may be followed.
[ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
[ ] c. Make no announcement or distrubution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear
Professor and Principal Investigator

Organization
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Signature Charles W Gear.

Date December 1978

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

[ ] a. AEC patent clearance has been granted by responsible AEC patent group.
[ ] b. Report has been sent to responsible AEC patent group for clearance.
[ ] c. Patent clearance not required.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-78-921	2.	3. Recipient's Accession No.
4. Title and Subtitle SIMULATION OF CONTINUOUS NETWORKS WITH MODEL			5. Report Date December 1978
			6.
7. Author(s) Mitchell G. Roth, Thomas F. Runge			8. Performing Organization Rept. No. UIUCDCS-R-78-921
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No.
			11. Contract/Grant No. US ENERGY/EY-76-S-02-2383
12. Sponsoring Organization Name and Address U. S. Department of Energy Chicago Operations Office 9800 South Cass Avenue Argonne, Illinois 60439			13. Type of Report & Period Covered research
			14.

15. Supplementary Notes

16. Abstracts

This manual is intended for the student, scientist, or engineer whose interests involve the solution of ordinary differential equations. In many cases, such equations arise in connection with the development of computer simulation models. MODEL (Modular Ordinary Differential Equation Language) is a very general and easy to use package for the formulation and simulation of continuous system models based on ordinary differential and nondifferential equations and generalized network (i.e., connective) effects. Partial differential equations can also be handled when they are transformed to a system of ODEs by the finite element method or the method of lines. MODEL combines both equation-oriented and network-oriented modeling features with a stiffly-stable implicit integration method allowing it to be applied more widely and easily than any other continuous simulation package.

17. Key Words and Document Analysis. 17a. Descriptors

simulation language
ordinary differential equations
network simulation

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 160
	20. Security Class (This Page UNCLASSIFIED	22. Price